(2)

Report BCM-NS-CNS-91-002

A Study of Neuronal Properties, Synaptic Plasticity and Network Interactions
Using a Computer Reconstituted Neuronal Network Derived from Fundamental Biophysical Principles

David C. Tam

Division of Neuroscience
Baylor College of Medicine
1 Baylor Plaza
Houston, TX 77030

(713) 798-3134

E-mail address:
dtam@next-cns.neusc.bcm.tmc.edu or
dtam@cephalo.neusc.bcm.tmc.edu

December 15, 1991

Semiannual Performance (Technical) Progress Report (June, 1991 - December, 1991)

Grant number: N00014-90-J-1353

Prepared for
Office of Naval Research
Biological Intelligence Program
800 North Quincy Street
Arlington, VA 22217-5000

Progress Summary

A standalone executable program of the neural simulator *MacNeuron* is developed on the Macintosh IIci and Quadra 900 computers. Two versions are available: one runs on the Macintosh II series computer with the math coprocessor installed, the other runs without the math coprocessor. The version with the math coprocessor executes faster than the one without the coprocessor. The current version (version 0.3 release) allows users to build a compartmental neuron, simulate the generation of action potentials and record any variables for graphical display. The variables include voltage, time and ionic concentration, etc. Each of these variables can be plotted on the graphical charts.

A Tutorial Manual and a User's Guide Manual are written describing the simulator, its run-time environment and a guide to use the neural simulator program.

The current version of *MacNeuron* implemented the numerical algorithms for driving the neural simulator. It also included the plotting routines for displaying the simulation results. The current version uses a window-based user-interface for building a network of neuron. The building process of the simulation environment is provided by the menu-driven window user-interface environment. An built-in text editor is provided within the simulation program for entering the script-file in describing the run-time environment.

A patent has been filed on the neural network signal decoding technology. The invention is "*An Interspike Interval Decoding Neural Network.*" used to decode the time interval between firing of neurons from a serial representation to a parallel representation.

A new statistical multiple spike train analysis method has also been developed. The novel technique uses a vectorial measure to detect correlation between firing patterns of multiple neurons simultaneously. The new technique enables detection of correlated firing pattern between any number of neurons, not just limited by the conventional pair-wise correlation.

A new biological experimental setup is being set up currently to record from multiple neurons simultaneously. The experimental results will be incorporated with the simulation results to test the hypotheses.

Specific Program Progress

Two major progresses are made in the neural simulation program. First, the numerical stimulation algorithms of neurons composed of patches of membrane (using the compartmental model) are implemented. The numerical integration routines are main computational engine for solving the systems of differential equations describing the biophysics of the ionic channels and receptors. Second, the graphical plotting of the simulation results are implemented. Any variables describing the simulation parameters can be plotted, thus providing visualization of the simulation results.

(A) Numerical Simulation Environment

The numerical engine for driving the neural simulator is implemented. The numerical integration algorithms used in this version is the conjugate gradient Euler methods. The numerical algorithms are required to solve the system of differential equations governing the dynamics of the ionic conductances, which is the key components in the neural simulator. Since our programming design is object-oriented, other numerical integration methods can be incorporated into our simulation program easily. In fact, the user will be able to choose the specific numerical methods at will when other numerical algorithms are implemented in the program. This is the essential feature in our simulator that most other simulators do not provide. Thus, it allows for flexibility for the user to choose whether to maximize the speed or accuracy of the simulation runs. Starting and stopping of simulation is done by a click of a "button".

(B) Graphical Plotting Display of Simulation Results

In the current implementation, the user can choose any two variables pair in the simulation and plot them graphically on a chart. Any variables used in the simulation can be "observed", and subsequently plotted on a graph. This provides flexibility to the user, so that the user can choose any variables, such as voltage, ionic concentration, gating activation variables, conductance, time, etc. can be observed during the simulation. To provide an intuitive approach to the user in selecting the plotting variable, the user needs only to "drag" any variable from a window which displays the list of parameter and "drop" that variable into the "observer" window and the "plot" window for plotting. Plotting of the results is done by clicking the "plot" button.

Thus, the current version of *MacNeuron* provides some basic building block for constructing a neuron for simulation. The incorporation of voltage-dependent conductance ionic gating channels on membranes for simulation and generation of action potentials can be accomplished currently in the simulator. Voltage step stimulation can also be accomplished for experimental manipulation.

Our future plan is to further improve the user-interface to provide more intuitive approach to construct the simulation environment. Further efforts will be put into incorporating the "macro" script-language into the simulator, so that all iconic steps for constructing the simulation environment can be captured by the macro script language automatically by the program, which can be saved into a file for later use or modification.

Invention of Interspike Interval Decoding Neural Network

A neural network using pulse-coded signal for processing is developed to decode interspike interval between the firing of action potentials in neurons. The neural network uses time-delays and appropriate excitatory and inhibitory connections for signal processing. With appropriate connections using a cascaded time-delay (or time-shifting) scheme, the signals are able to propagate to different neurons appropriately by the exact time interval for extracting the firing intervals. This neural network represents the outputs in a two-dimensional topographical map configuration, where the location of neurons which fire at the output layer represents the firing interval being decoded. The network, thus, able to extract serial-code into parallel-code. That is, it converts from serial representation of firing interval to parallel representation of firing by the location of the activated neurons.

Invention of a New Statistical Method for Detecting Correlation of Firing between any Large Numbers of Neurons

A novel statistical method is developed specifically for detecting the temporal correlation among firing of neurons. The new techniques uses a vectorial measure to represent the preceding and succeeding firing interval between any neurons. A vector is used to represent this preceding and succeeding firing intervals, called cross-intervals. Thus, this cross-interval vector represents the timing relationships between the firings of any neurons. To compute the statistical average of the population, the resultant vector, which is the vectorial sum of the population, can be used to represent the present the timing relationship of all neurons relative to

the reference neuron. Thus, this vectorial measure is used to detect any correlational relationship between any reference neuron and the rest of the population.

Biological Experimental Preparation Setup

A new biological experimental setup is being set up currently to perform experiments on biological neural preparation. The multi-electrode recording setup will have at least 32 channels recording capability. The system is capable of expanding to record from 256 channels simultaneously. Two types of electrodes will be used. A 64-channel microelectrode photo-etched on transparent glass plate will be used to record from neuronal cell culture. A neural network can be grown on such electrode plate, which can then be recorded and stimulated simultaneously. Since the electrode is also transparent, optical imaging recording can be done simultaneously with the electrical recording. Such recording can be done *in vitro* on cell culture or brain slice.

Another setup will also be used for recording *in vivo*. A multi-stranded electrode bundle will be used to record from the cortices of animal implanted with this electrode. Thus psychophysical experiments on learning and memory can be done while recording from the activity of the neurons in the cortical network.

Patent Pending

Tam, D. C. (Dec., 1990) *An Interspike Interval Decoding Neural Network.* Serial No. 07/630,463.

Publications by the Principal Investigator during 1990 and 1991

Tam, D. C. (1991) Signal processing in multi-threshold neurons. In: *Single Neuron Computation* (T. McKenna, J. Davis, and S. F. Zornetzer, eds.) Academic Press, San Diego. pp. 481-501.

Tam, D. C. (1991) Signal processing by multiplexing and demultiplexing in neurons. In: *Advances in Neural Information Processing Systems.* (D. S. Touretzky, ed.), Morgan Kaufmann Publishers, San Mateo, California. pp. 282-288.

Tam, D. C. (1991) A hybrid time-shifted neural network for analyzing biological neuronal spike trains. *Progress in Neural Networks* (O. Omidvar, ed.)Vol. 2, Ablex Publishing Corporation: Norwood, New Jersey. (*in press*)

Alkon, D. L., Vogl, T. P, Blackwell, K. T. and Tam, D. C. (1991) Memory function in neural and artificial networks. In: *Neural Network Models of Conditioning and Action.* (M. L. Commons, S. Grossberg, J. E. R. Staddon, eds.) pp. 1-11. Lawrence Erlbaum Associates: Hillsdale, New Jersey.

Tam, D. C. (1990) Decoding of firing intervals in a temporal-coded spike train using a topographically mapped neural network. *Proceedings of the International Joint Conference on Neural Networks, June, 1990.* Vol. 3, pp. III-627–632.

Tam, D. C. (1990) Temporal-spatial coding transformation: Conversion of frequency-code to place-code via a time-delayed neural network. *Proceedings of the International Joint Conference on Neural Networks* (H. Caudill, eds.), *Jan., 1990.* Vol. 1, pp. I-130–133.

Abstracts

Tam, D. C. and Kenyon, G. T. (1991) A vectorial statistical method for detecting correlated firing patterns in neurons. *Biophysical Society Abstract.* (in press)

Kenyon, G. T. and Tam, D. C.(1991) An object-oriented paradigm for simulating physiological processes in extended biological structures. *Biophysical Society Abstract.* (in press)

Tam, D. C. and Kenyon, G. T. (1991) A novel vectorial measure for detecting temporally correlated firing patterns in multiple spike trains. *Society for Neuroscience Abstract.* Vol. 17, p. 125.

Boney, D. G., Feinswog, L. J., Hutson, R. K., Kenyon, G. T. and Tam, D. C. (1991) An object-oriented paradigm for simulating inter-connected neural systems. *Society for Neuroscience Abstract.* Vol. 17, p. 126.

Tam, D. C. (1991) A vectorial statistical method for analyzing stochastic firing patterns in large numbers of neurons in parallel. *The Second Keck Symposium on Computational Biology* (in press)

Feinswog, L. J., Hutson, R. K., Kenyon, G. T. and Tam, D. C. (1991) An object-oriented paradigm for simulating neurophysiological processes. *The Second Keck Symposium on Computational Biology* (in press)

Tam, D. C. (1990) Functional significance of bi-threshold firing of neurons. *Society for Neuroscience Abstract.* Vol. 16, p. 1091.

Tam, D. C. (1990) Hebbian synapse and its relation to cross-correlation function in associative conditioning learning. *Eighth Annual Conference on Biomedical Engineering Research in Houston.* p. 5.

Tam, D. C. (1990) Visual-motor integration: What role does cerebellar cortical neurons participate in movement control in monkeys? *Eighth Annual Conference on Biomedical Engineering Research in Houston.* p. 39.

MacNeuron (version 0.2.3) User's Guide

1. Startup MacNeuron program

The *MacNeuron* (version 0.2.3) program's icon looks like this:

MacNeuron v0.2.3

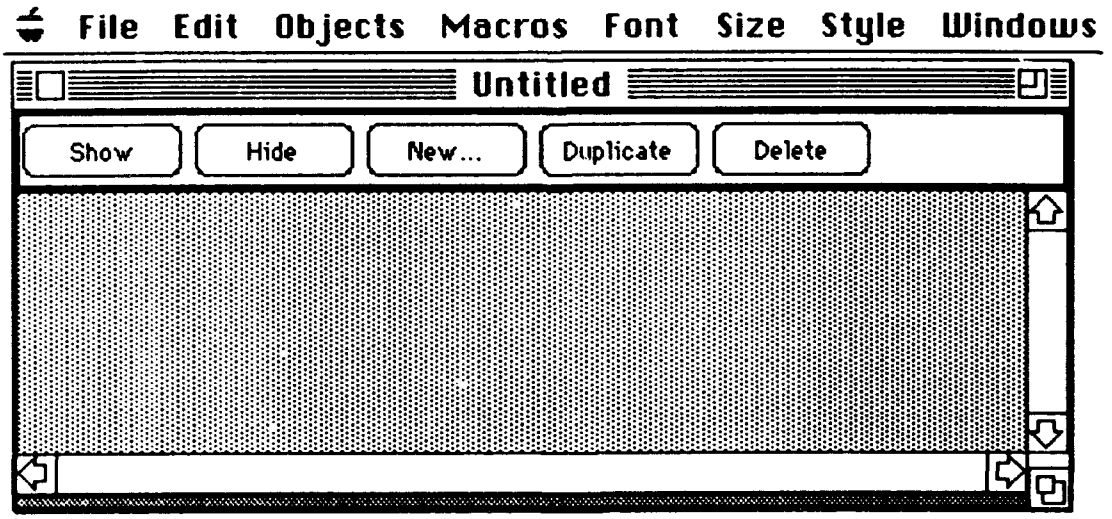
Start executing (launching) the *MacNeuron* (version 0.2.3) program by selecting the program icon:

MacNeuron v0.0.3

Launch the *MacNeuron* application program as usual in the Macintosh environment by double-clicking on the icon or selecting the "*Open...*" menu-item from the "*File*" menu.

2. Main MacNeuron Window

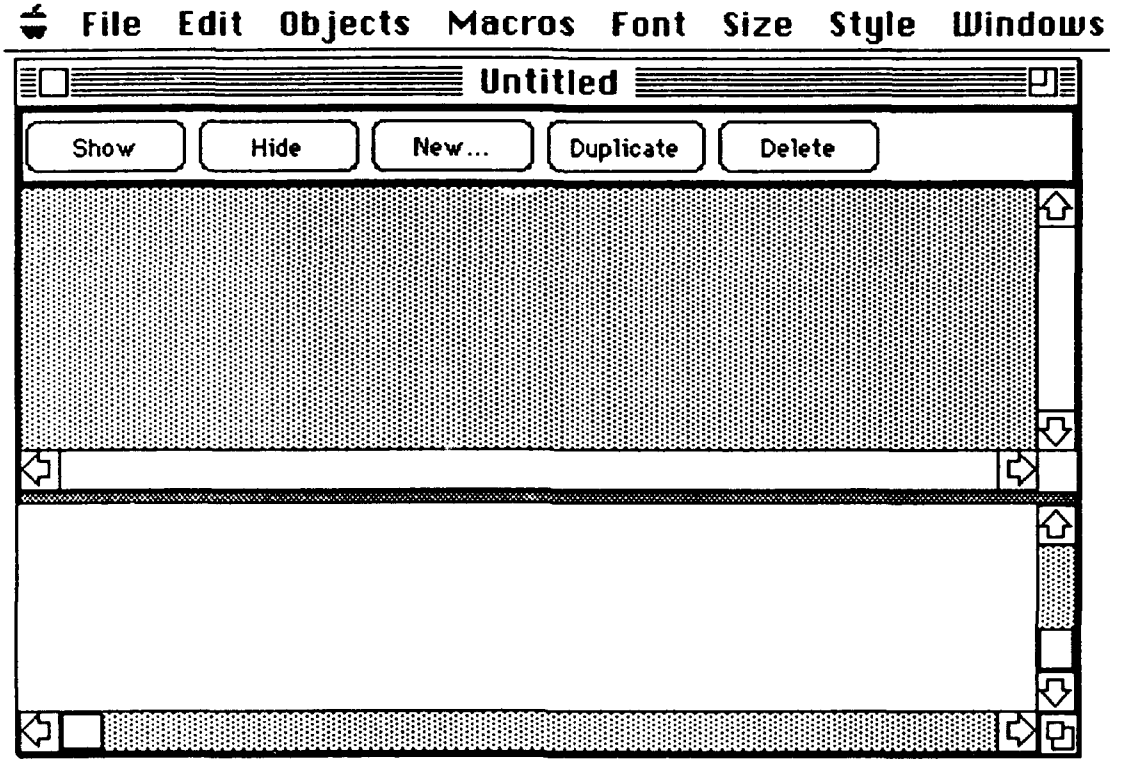
When the program first starts looks like the following :



The program will startup with a "*New*" window called "*Untitled*". This is the *main window* of the application. The name of this main window will change when the file is saved, just like any other standard Macintosh program.

2.1. Resizing the Main MacNeuron Window Partitions

This main window contains two parts (or partitions). When the resize box at the lower right corner of the window is enlarged, the second half (partition) of the window will show up. The window will appear as follows:



Alternatively, one can hold-down the mouse button at the window-partition divider (the shaded thick line between the two partitions) and "drag" the divider up or down to change the size of the partition. Initially, the partition divider is at the bottom of the window, but it can be dragged up to show the lower window partition.

2.2. Neuronal Network Description. User-Interface Window Partitions

The upper partition of this main window lists all the objects (neurons, conductances, compartments, networks, etc...) currently in the simulation. The upper portion of the window also provides a palette of tools for performing useful operations on the listed items. For instance, selecting a particular neuron from the list and then clicking the show button will cause the window for that neuron to be displayed. There are also tools for hiding, deleting and duplicating any of the objects in the list. Procedures for building the simulation will be described later in User's Guide.

2.3. Neuronal Simulation Language Description. Text-Window Partitions

The lower partition of the main window is the text window where the simulation language is specified. A text-based language, called *"macro"*, can be used to specify the simulation without recourse to the iconic user-interface. This allows the simulation to be built in a "batch-mode" or hands-off modality.

This window is also a text-editor, where the macro language description can be entered directly into the *MacNeuron* program. It is not necessary to exit the *MacNeuron* application to write the description.

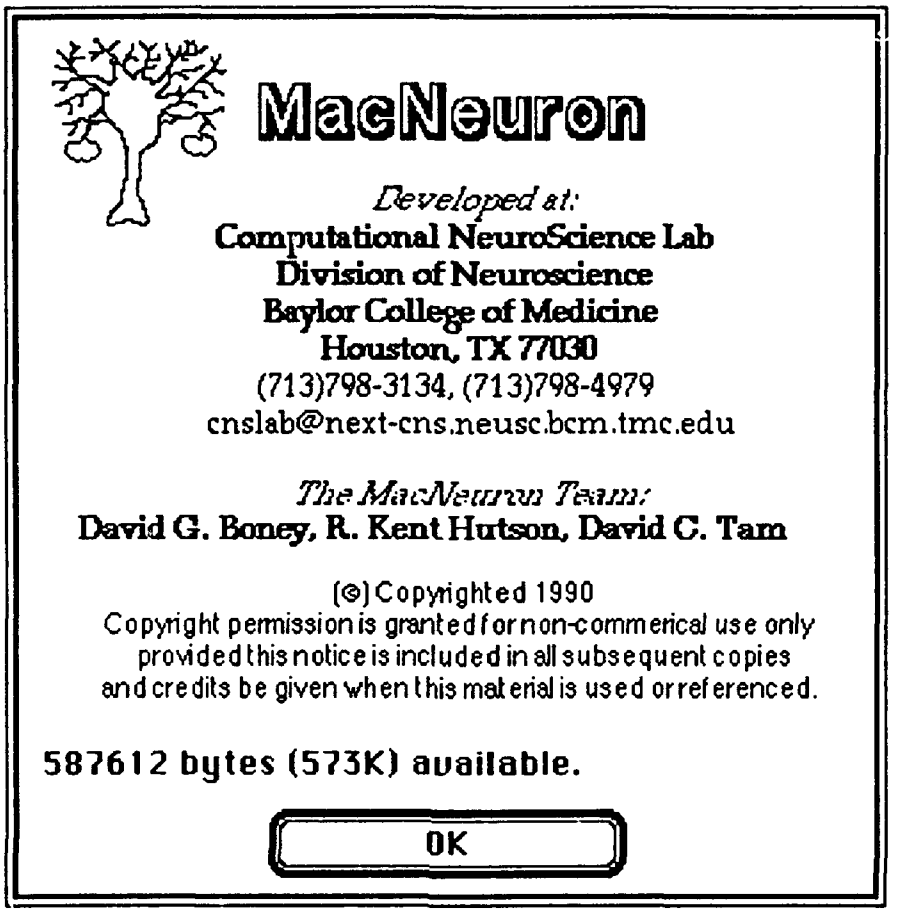
The details of the macro language syntax for specifying the run-time environment will be given later in the User's Guide.

3. Main Menu Description

3.1. *Apple* Menu Description

Pulling-down the Menu will show the *"About MacNeuron..."* as the first item.

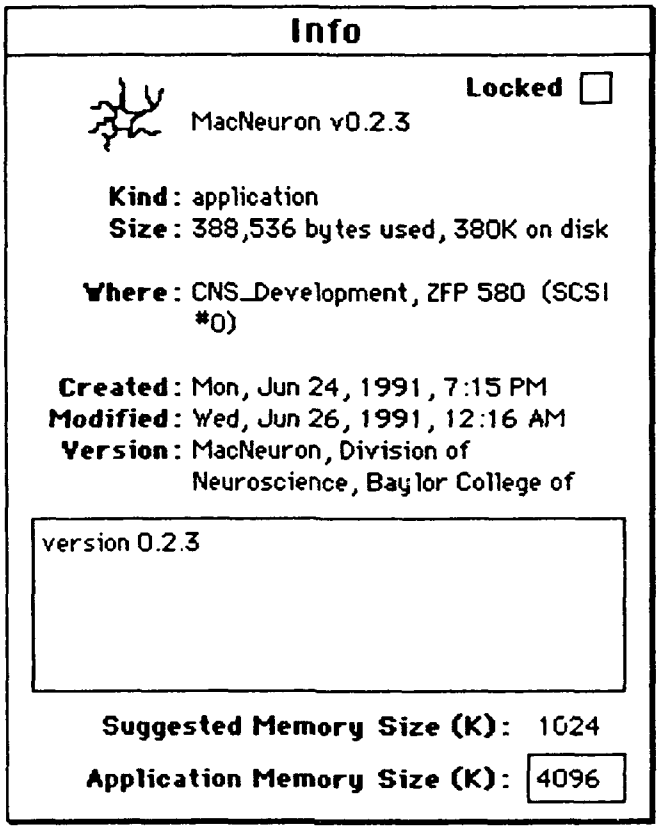
The *"About MacNeuron..."* will display the program logo as well as the available memory space for the program to run at the bottom of the logo window.



3.1.1. Memory Space Available and Allocation

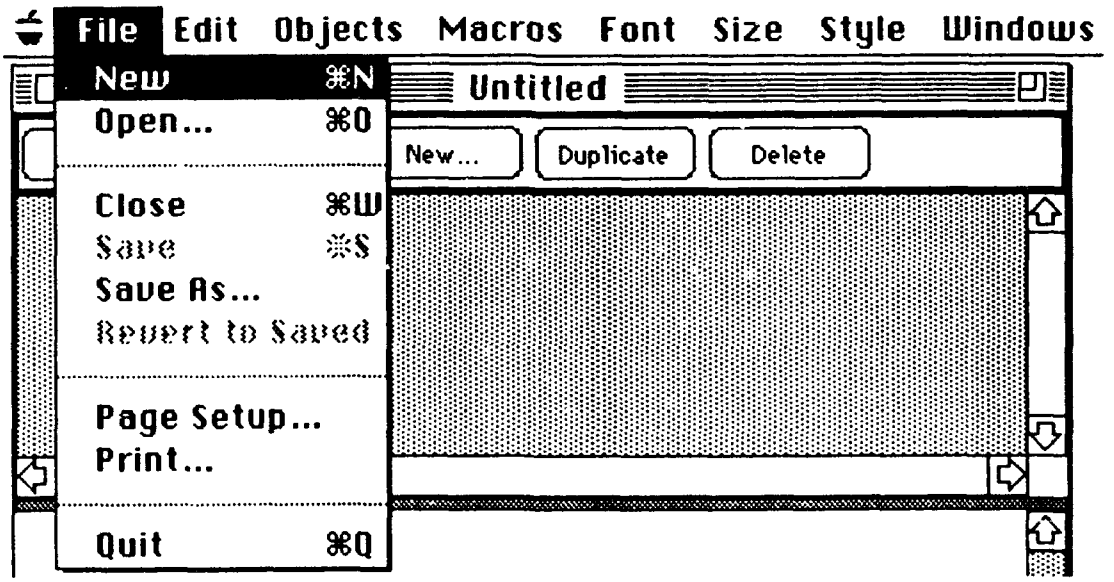
To find out how much memory is available at anytime while inside the simulator environment, select the "*About MacNeuron...*" from the *Apple* Menu as shown above. The memory available depends on the complexity of the neuronal network specified and other run-time user-interface environments.

To increase or decrease the size of the memory allocation, quit the program. Select the *MacNeuron* program icon under *Finder*, and select the "*Get Info*" menu-item from the *File* Menu. An *Info* window similar to the one below will be displayed.



Change the size of the *MacNeuron* program by selecting the *Application Memory Size (K)* box and edit the changes in Kilobytes.

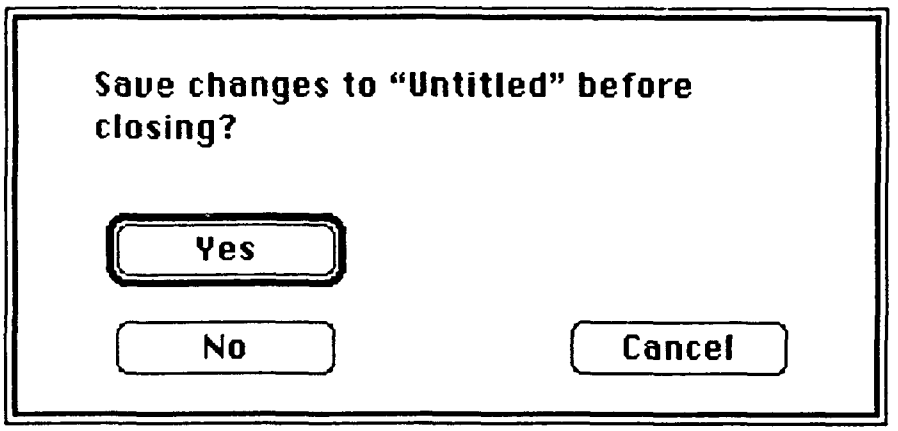
3.2. *File* Menu Description



The *File* Menu has its usual standard Macintosh-menu items for file operations such as opening and closing files, saving and printing files, creating new files, quitting the application, etc.

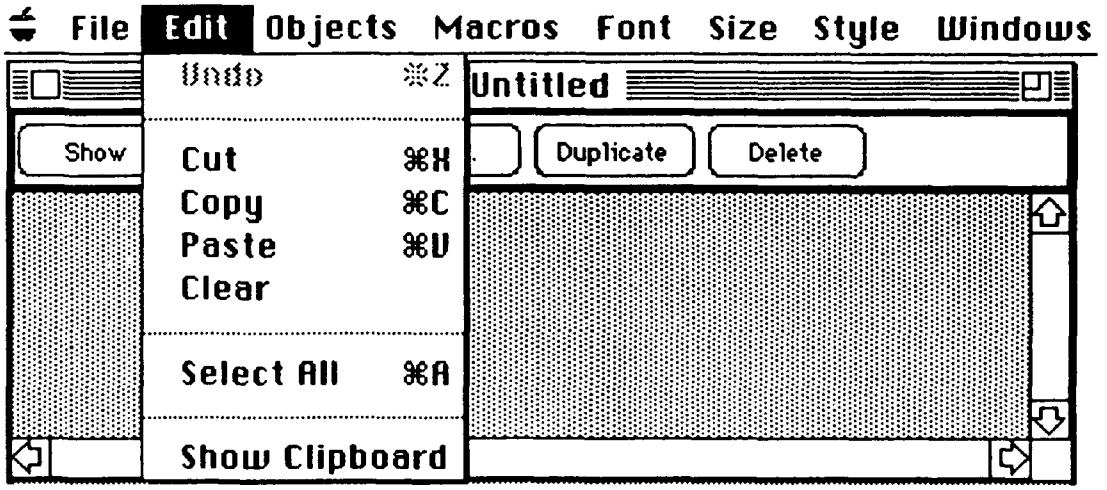
When the program is first launched, a new file is created automatically for the user and appears as the "*Untitled*" main *MacNeuron* description window. This would also be the consequence of clicking the "*New*" menu-item in the *File* Menu.

To close this main window, select the "*Close*" menu-item or click on the *close*-box of the window at the upper-left corner of the window as usual. A warning dialog-box will be displayed is you have not saved the file before to make sure that you don't loose any valuable work.



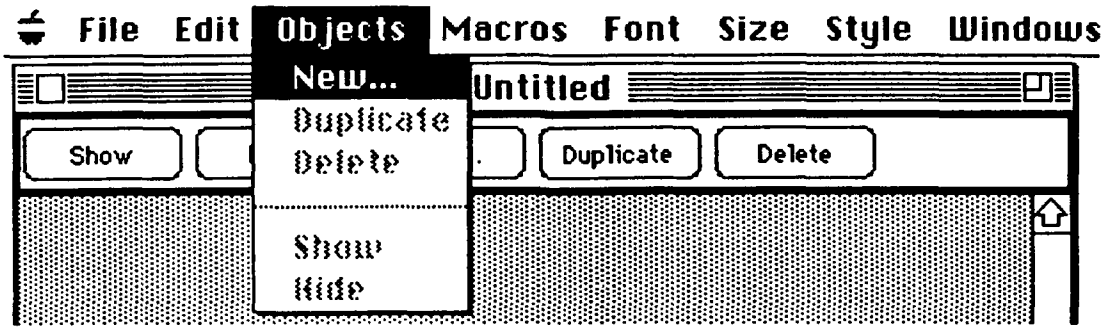
Select the appropriate button or press the `Return`-key of the keyboard to select the highlighted button, which is "*Yes*" in this case.

3.3. *Edit* Menu Description



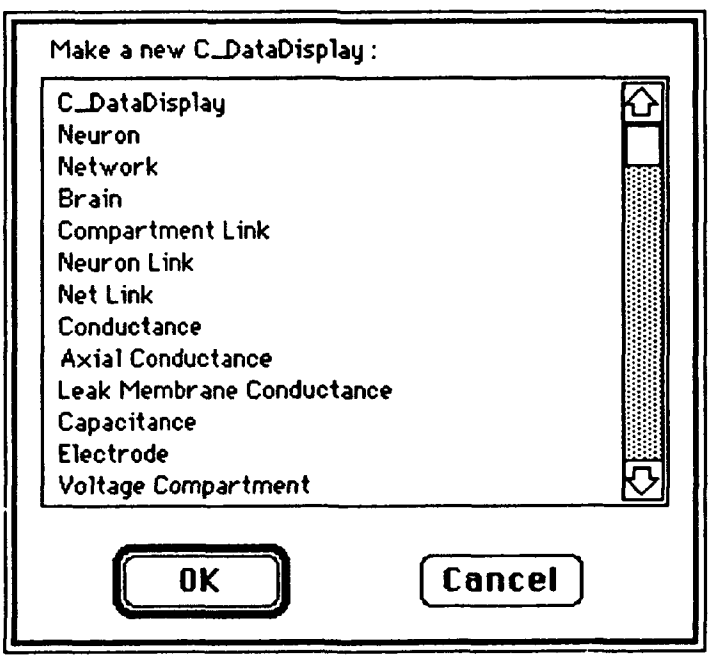
The *Edit* Menu has its usual standard Macintosh-menu items for editing. The *Cut, Copy, Paste* and *Clear* menu items can be used with the text window for editing the simulation macro description language (which is a lower sub-window in the main window).

3.4. *Objects* Menu Description



Objects in the *MacNeuron* program are the implementations of the components of a neuron or a network. The currently available objects in *MacNeuron* include *brain, network, neuron, voltage compartment, conductance, axial conductance, membrane conductance, leakage conductance, capacitance, electrode, active conductance, HH gate* (Hodgkin-Huxley), *compartment link, neuron link, network link,* etc. as well as objects for displaying data.

A list of the available objects will be displayed in a dialog box when the "*New...*" menu item from the *Objects* Menu is selected.



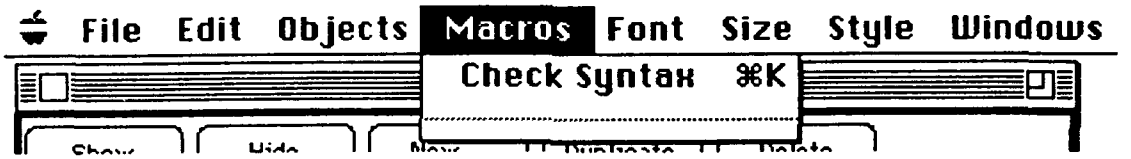
The other menu items, such as *Duplicate, Delete, Show* and *Hide* are dimmed initially since no object exists until a new objects is created first.

3.4.1. Creating New *Objects* from the *Objects* Menu

To create new objects, select the desired object in the above dialog box, and click the "*OK*" button. Alternatively, you can double-click on the selected object without clicking on the "*OK*" button to create that new object. To abort the creation of a new objects, click the "*Cancel*" button.

Once the new object is created or "*New*ed", a new window associated with that new object will pop up. In the above example, since the *brain* object is selected for creation, the *brain* window will pop up as the front window. The detailed descriptions and utilities of this new object window will be discussed later in this User's Guide.

3.5. *Macros* Menu Description



Macros are groups of commands specified by the user in the simulation macro description language. The simulation description language—*MacNeuron Script*—is a Pascal-like language that allows a hands-off specification of the simulation bypassing the iconic user-interface. Such a feature is very useful for constructing large simulations intended to run in a batch or background mode. Details of the language syntax will be given later in the User's Guide.

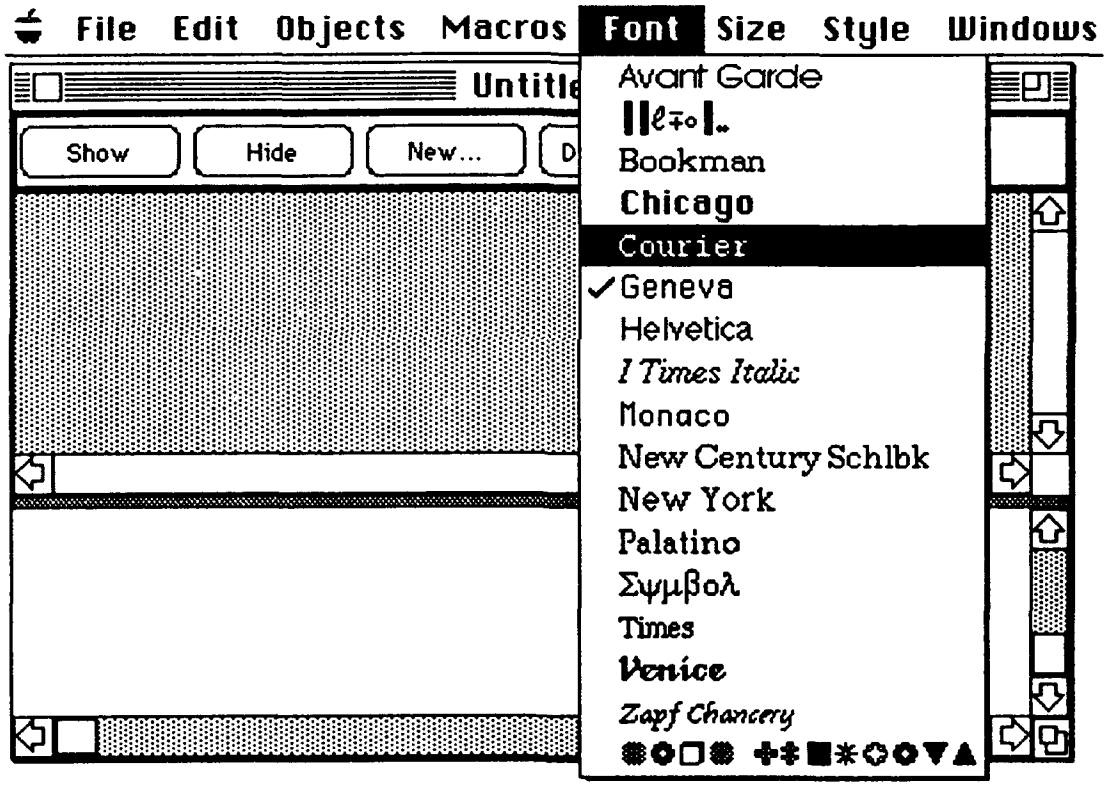
If the text of *MacNeuron Script* is entered in the text-editor window at the lower partition of the main window, selecting the "*Check Syntax*" menu item from the *Macros* Menu will check the syntax of the *MacNeuron Script* entered by the user. If there are any syntax errors, the appropriate error message will be displayed in a warning dialog box. The location of the offending error will be highlighted in the text-editor window to allow for easy recognition by the user.

Once the syntax has been checked, and there are no syntax errors, the *macros* defined in the *MacNeuron Script* will be displayed as extra menu items appended to the end of the *Macros* Menu lists. *Macros* are basically groups of commands that the user specifies in the *MacNeuron Script* so that those groups of commands can be

executed as a menu command available in the menu-item list. The user can therefore create his/her own menu-item list on-the-fly within the *MacNeuron* user environment.

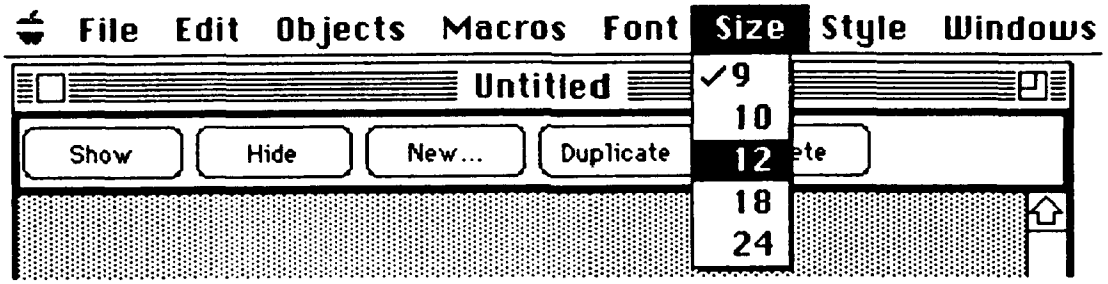
For instance, if the user groups a set of commands that specify the description of all the compartments of a Purkinje neuron into a single *macros* command called "*Create Purkinje Cell*", then a Purkinje cell can be created with just a mouse click from the *Macros* Menu. Thus, multiple Purkinje cells can be created easily by "pulling down" the *Macros* Menu and selecting the "*Create Purkinje Cell*" menu item that was defined by the user in the *MacNeuron Script* in the text-editor window.

3.6. *Font* Menu Description



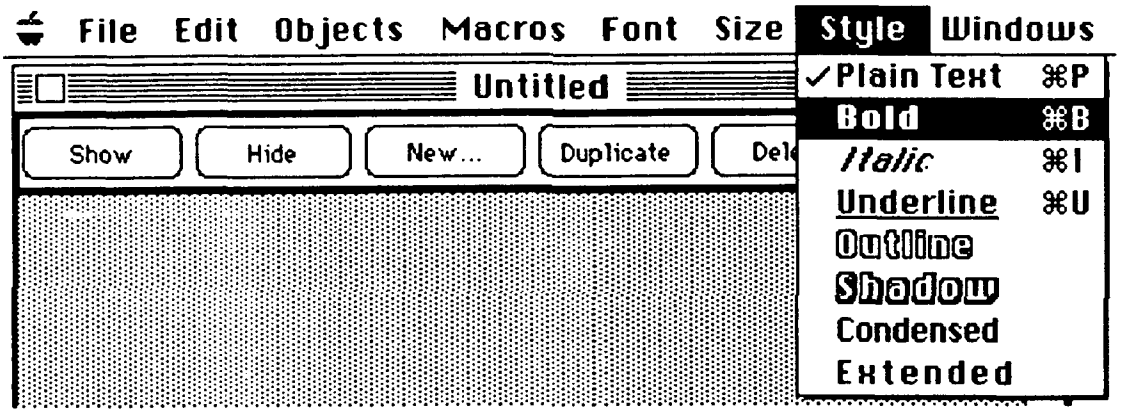
The *Font* selected by this menu will alter the font used in the text-editor sub-window of the main *MacNeuron* window. The available fonts are the fonts that are installed in the System file of your Macintosh disk. The actual font-type displayed in the *Font* Menu above is generated by a utility program called Suitcase™ installed in the System, otherwise the system-font will be used to display the various fonts available. *Geneva* is the default font.

3.7. *Size* Menu Description



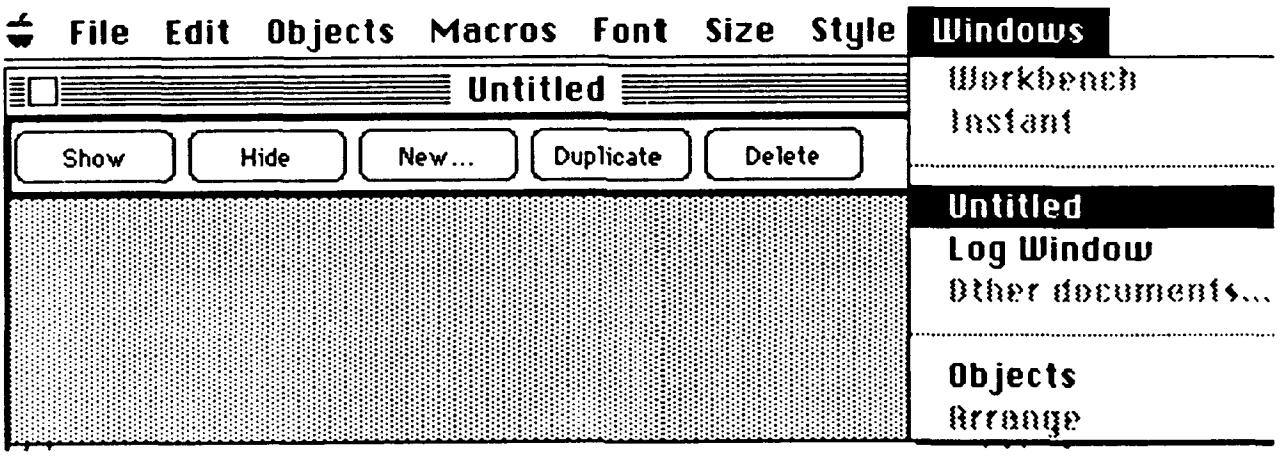
The *Size* selected by this menu will alter the font size used in the text-editor sub-window of the main *MacNeuron* window. The default font size is 9 points.

3.8. *Style* Menu Description



The *Style* selected by this menu will alter the font style used in the text-editor sub-window of the main *MacNeuron* window. *Plain-Text* is the default style.

3.9. *Windows* Menu Description



The *Windows* Menu lists all the available windows, and allows all of them to be selected as the active window (the front window).

3.9.1. *Main Window* Menu-Item

The "*Untitled*" window is the main *MacNeuron* window if it has not been saved before. If an existing file is opened, the name of that file is displayed in that menu-item location. Selecting it will make that window active (it will appear as the front window).

3.9.2. *Log Window* Menu-Item

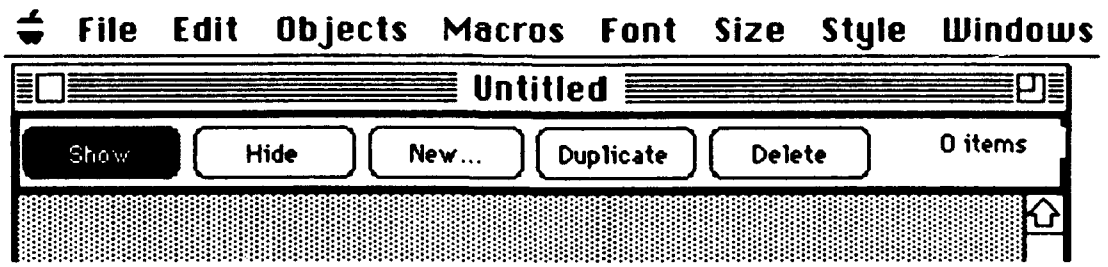
The *Log Window* is a text display window where the program keeps a log of the history of the simulation. The user can also select specific parameters to be output into this *Log Window*. Selecting it will make that window active (it will appear as the front window).

If the program is run in unattended batch-mode, the specified simulation textual output and any error messages will be displayed in this *Log Window*. If there are critical errors that require user's intervention (such as File-Not-Found) while running in batch-mode, the program will wait for a "time-out" period. If no user's action is taken after the time-out, a default value (such as a default file-name) will be used. This will allow for unattended continued simulation over-night without creating halting the program waiting for the user's response.

3.9.3. *Object Window* Menu-Item

The *Object Window* is a window where the contents (and the parameters) of an *object* are displayed. Selecting it will make that window active (it will appear as the front window). For instance, if the object is a neuron, then the contents of that neuron, i.e., its parameters will be displayed in this object window.

4. *Main MacNeuron User-Interface Description Window*



The main *MacNeuron* user-interface description window (the upper partition of the Main window) contains five buttons. These five buttons are the same as the menu items listed under the *Object* Menu (see *Object* Menu Description above). Their use is interchangeable. That is, the buttons function the same as the menu items in the *Object* Menu.

4.1. *Show* Button

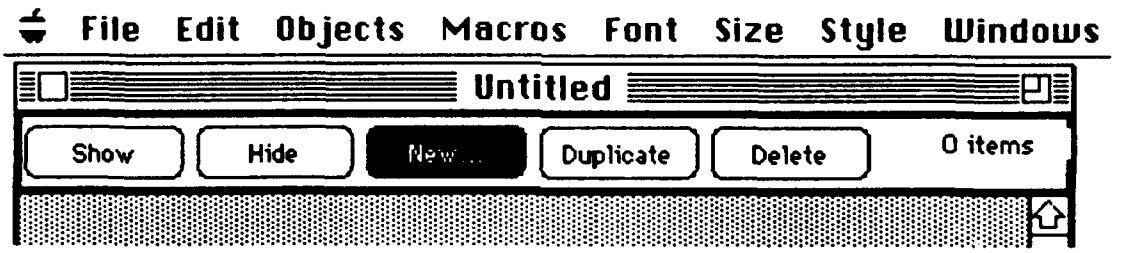
When the "*Show*" button is pressed, it will show the object that is selected in a separate *Object Window*. Since, at this point, no objects are created yet, there is nothing to show. The "*0 items*" is indicated at the upper-right of the window to show the number of object-items created so far.

Once objects are created, they will be listed in the shaded region of the window. Select the object by clicking on the item in the list, and click the "*Show*" button. The *Object Window* associated with that selected object will pop up (open). The contents of that object (i.e., its parameters) will be displayed in the *Object Window*.

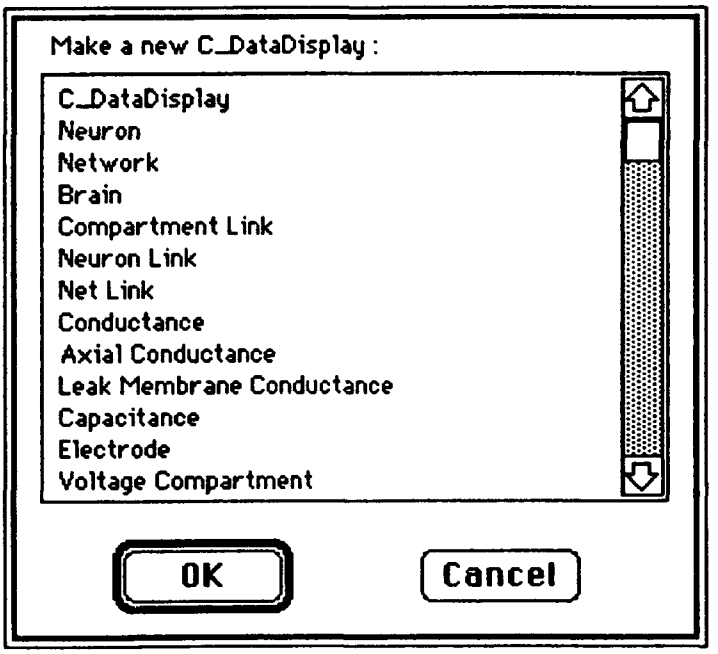
4.2. *Hide* Button

When the "*Hide*" button is pressed, it will hide (close) the *Object Window*. Which *Object Window* will be closed depends upon the selected objects in the object list displayed in this window.

4.3. *New...* Button



When the "*New...*" button is pressed, a list of the available objects will be displayed in a dialog box. Select the object to be created as described above (see the Creating New *Objects* from the *Objects* Menu Section).



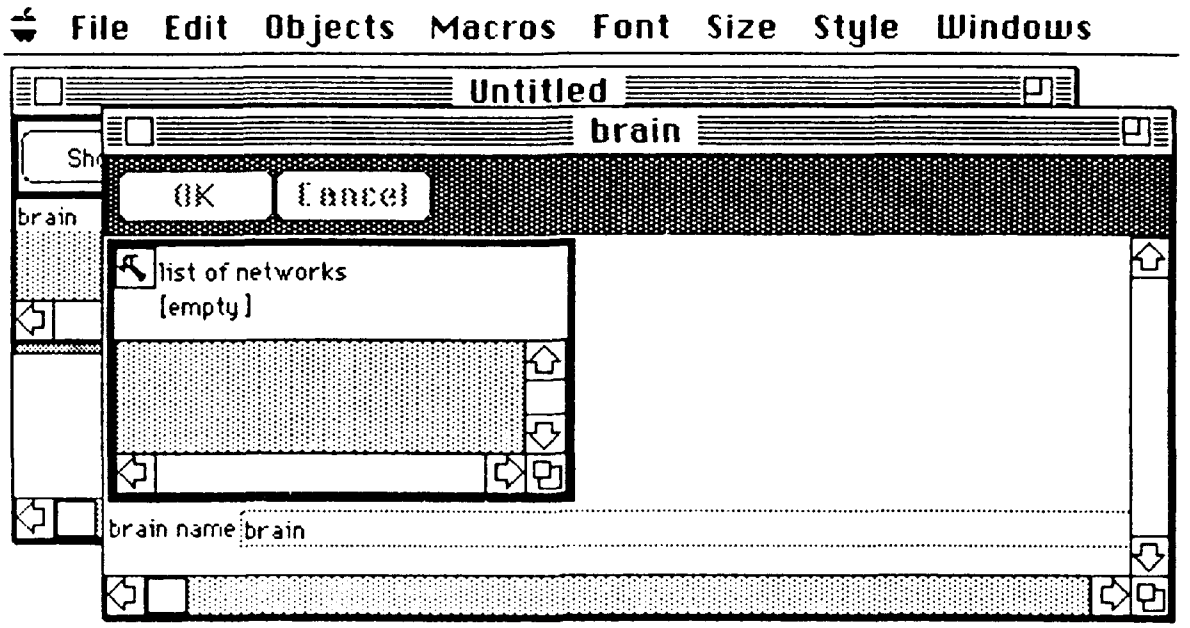
4.4. *Duplicate* Button

When the "*Duplicate*" button is pressed, the selected objects in the object list displayed in this window will be duplicated.

4.5. *Delete* Button

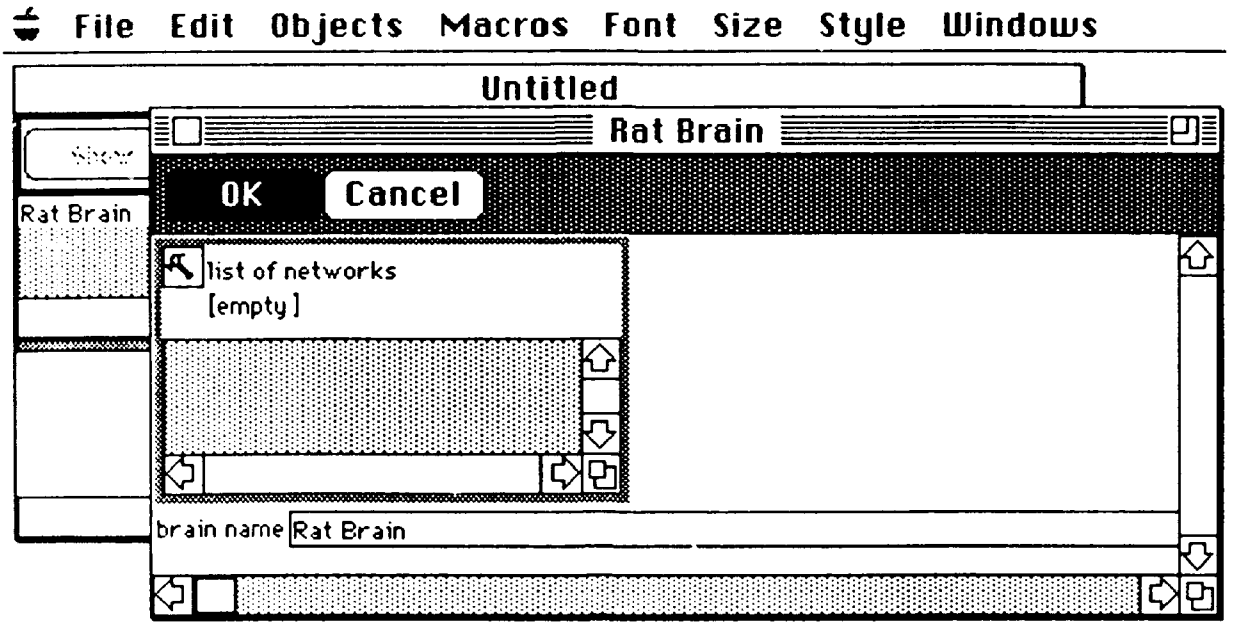
When the "*Delete*" button is pressed, the selected objects in the object list displayed in this window will be deleted.

5. *Brain Object* Window



The *Brain Object* window is a window where the contents (and parameters) of the *brain* are displayed. Selecting it will make that window active (it will appear as the front window). For instance, since in this case the object is a *brain*, the content of that *brain*, i.e., its parameters, will be displayed in this object window. The *title* of this *Brain Object* window will be called by the name of the object, i.e., the *brain* by default. The name of the *Brain Object* can be changed, as it will be discussed later.

5.1. *List of Network Objects* Window



There are usually two sets of items to be displayed in a *Brain Object* window. The first set of items is the "list of other objects under its hierarchy". In this example, the *brain* is composed of a list of *networks*. Similarly, a *network* is composed of a list of *neurons*. A *neuron* is composed of a list of *compartments*, etc. Thus, the anatomical structure of a brain, a network, a neuron, etc., can be specified hierarchically.

In this example, since the *brain* has just been created, the list of *networks* is not specified yet. So "*[empty]*" is indicated in the "*list of networks*" sub-window.

Note that the "*list of networks*" sub-window is highlighted (i.e., a thick dark square outlines the window. This is the item which is currently selected. You can change the selected item by pressing the "tab" key in the keyboard to "tab over" to the next field as in any Macintosh application environment. The next item is the "*brain name*" which will be highlighted.

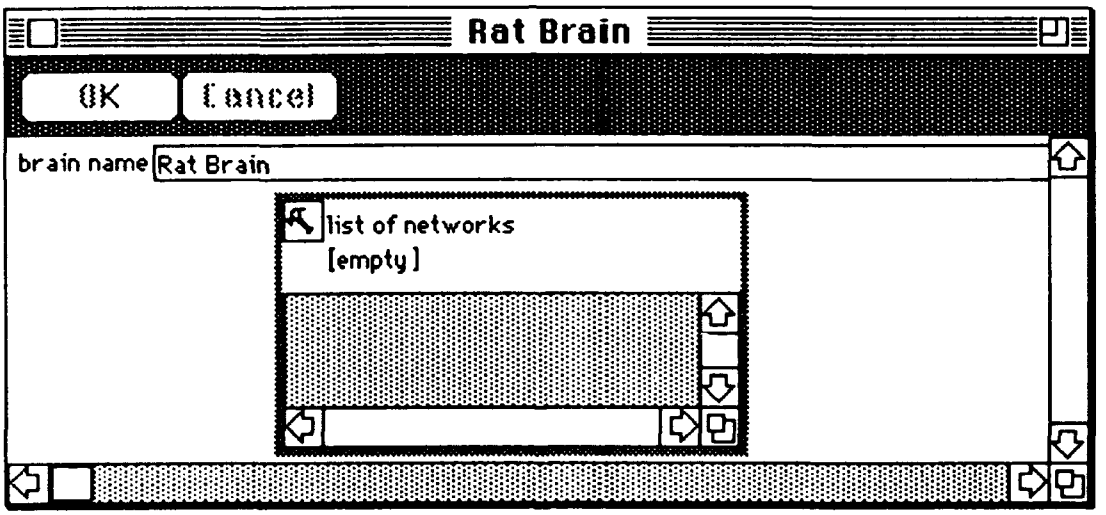
Alternatively, you can use the mouse to move the pointer over the "*brain name*" box and edit the text as usual. The "*brain name*" box will be selected (or highlighted), and the text can be changed accordingly.

In the example shown above, the name is changed into "*Rat Brain*". Press the "*OK*" button to confirm the changes or press the "*Cancel*" button to cancel and revert the changes. When the "*OK*" button is press, the new name "*Rat Brain*" will be

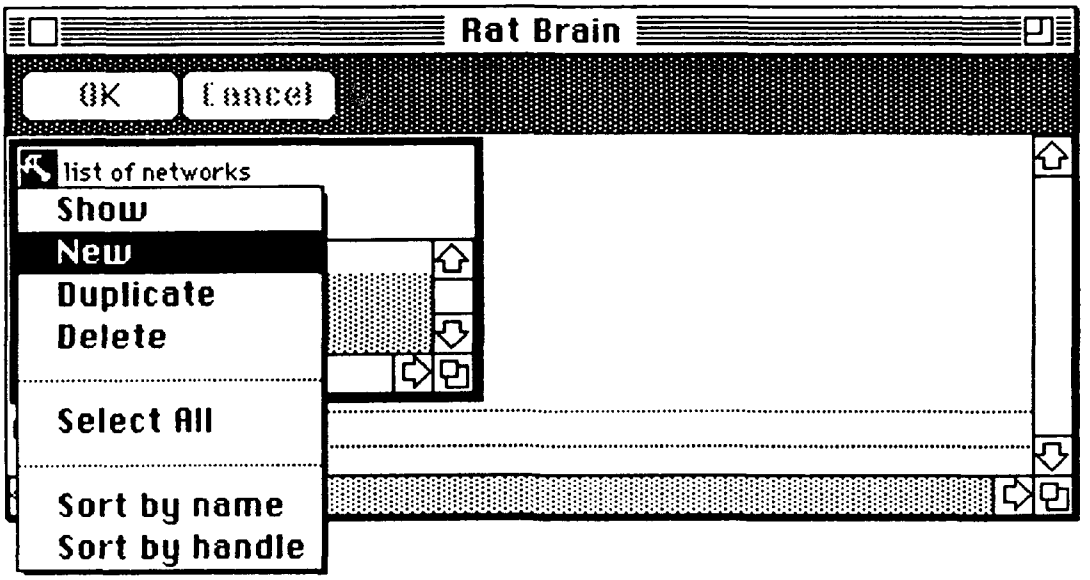
reflected in the title bar of the current "*Brain*" window as the new "*Rat Brain*" window. This new brain name is also reflected in the item list of the main window, which is still called "*Untitled*" presently (half-hidden behind the "*Rat Brain*" window in the above example).

Note that the appearance of the layout arrangement of the selectable items such as the "*list of networks*" sub-window and the "*brain name*" box can be rearranged by the user. Holding down the `command`-key (or the "`apple-clover`" key) of the Macintosh keyboard while pressing the mouse button over the selected item will change the cursor into a four-arrow sign (✣), indicating that the user can now move that item to a new location within the window. This four-arrow sign will also appear when the cursor is placed above the name field of the "*list of networks*" sub-window without holding down the `command` key.

For instance, the layout of the "*Rat Brain*" window can be re-arranged to look like the following by the user:



5.1.1. Pop-up Menu in the *List of Networks* Window



When the *tool* icon ( ) at the upper-left corner of the *List of Objects* sub-window is pressed, a pop-up menu will appear as described below.

5.1.2. *Show* Menu-Item from the Pop-up Menu

The *Show* Menu-Item shows the selected network from the list of networks created so far. A *Network Object* window will pop-up displaying the contents (parameters) of the network.

Alternatively, the selected item (from the lists of networks) can be double-clicked to show the *Network Object* window.

5.1.3. *New* Menu Item from the Pop-up Menu

The *New* Menu Item creates a new network and numbers it sequentially with an integer number in parenthesis. The new network will be displayed in the shaded region of the list of networks window.

5.1.4. *Duplicate* Menu Item from the Pop-up Menu

The *Duplicate* Menu Item clones a new network and numbers the newly cloned network sequentially with an integer number in parenthesis. The new network will be displayed in the shaded region of the list of networks window.

5.1.5. *Delete* Menu Item from the Pop-up Menu

The *Delete* Menu Item removes a selected existing network from the list of networks displayed in the shaded region of the list of networks window.

5.1.6. *Select All* Menu Item from the Pop-up Menu

The *Select All* Menu Item selects all the existing networks from the list of networks displayed in the shaded region of the list of networks window.

5.1.7. *Sort by name* Menu Item from the Pop-up Menu

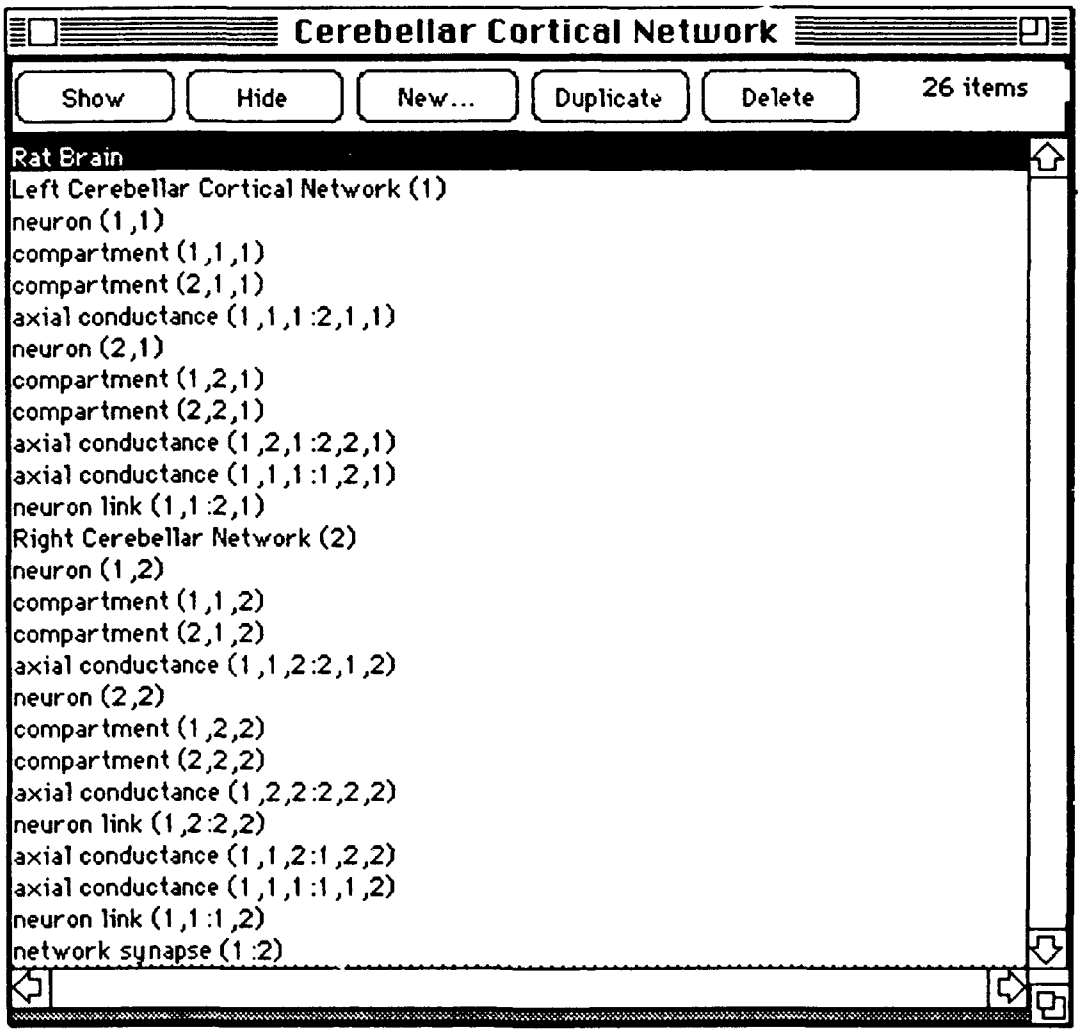
The *Sort by name* Menu Item re-orders the existing networks in alphabetical order using the name of the network displayed in the shaded region of the list of networks window.

5.1.8. *Sort by handle* Menu Item from the Pop-up Menu

The *Sort by handle* Menu Item re-orders the existing networks in internal order managed by the computer of the network (called the handle to the network).

6. Cerebellar Cortical Network Example

We will use the pre-built Cerebellar Cortical Network file as an example.

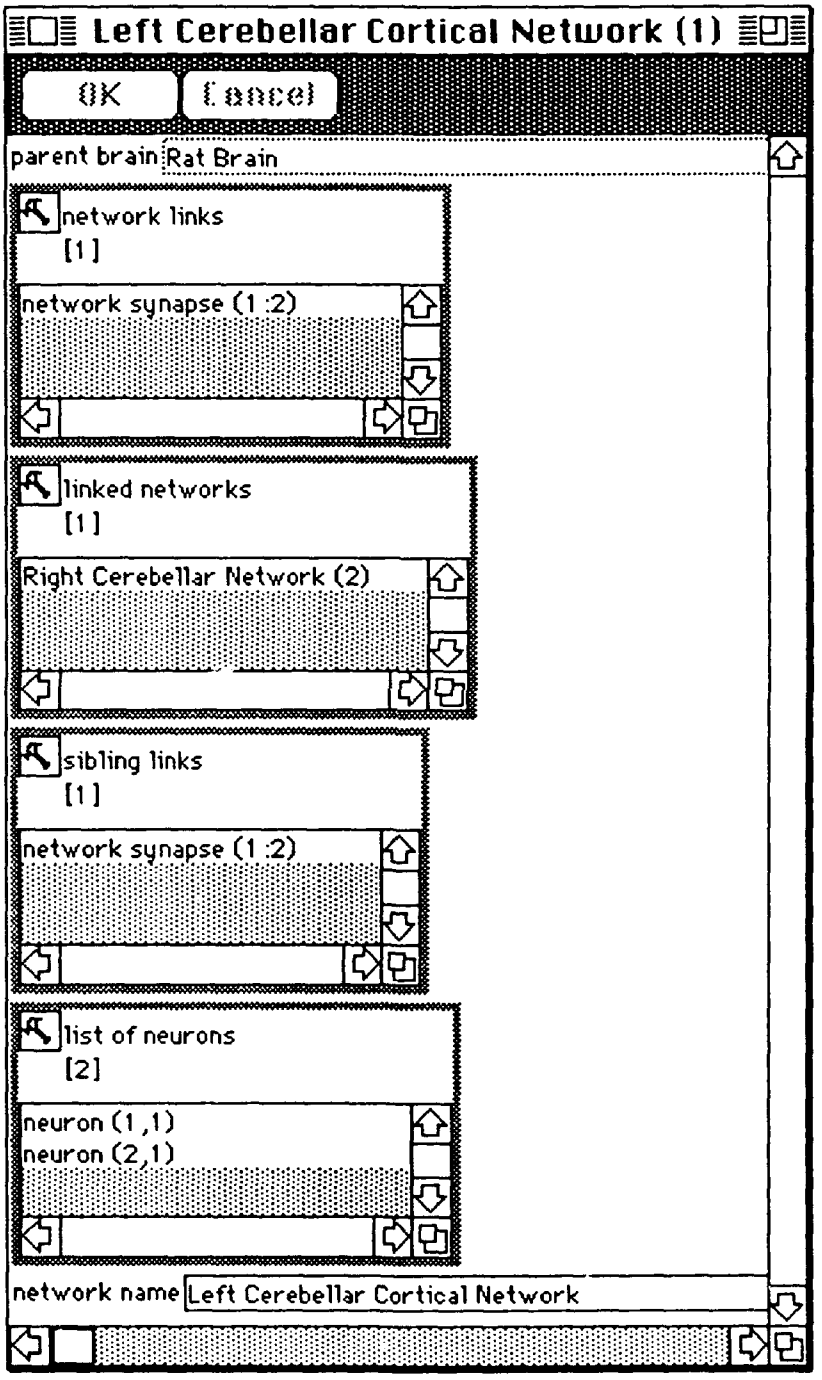


7. *Network Object* Window

The *Network Object* window is a window where the contents (and parameters) of the *network* are displayed. Selecting it will make that window active (it will appear as the front window). For instance, since in this case the object is a *network*, the content of that *network*, i.e., its parameters will be displayed in this object window. The *title* of this *Network Object* window will be the name of the object, i.e., the *network (1)* by default. The name of the *Network Object* can be changed, as it will be discussed later.

Since the simulated neural network is constructed hierarchically, we will use family-tree terminology to refer to the hierarchical structure, such as parent object, sibling links, etc.

In this example, the parent object of this newly created network is the *Rat Brain*. The name of this network can be changed to another name, say *Cerebellar Cortical Network*.



7.1. *Network Links* Sub-Window

This shows the list of network synapses to which the current network is connected. The numeric indices in parentheses refer to the network and the network synapse by their number (i.e., their name for identity). It shows that *Left Cerebellar Network (1)* is connected to the *Right Cerebellar Network (2)* by specifying *Network Synapse (1:2)*.

7.2. *Linked Networks* Sub-Window

This shows the list of connected networks. It is connected to the *Right Cerebellar Network (2)*.

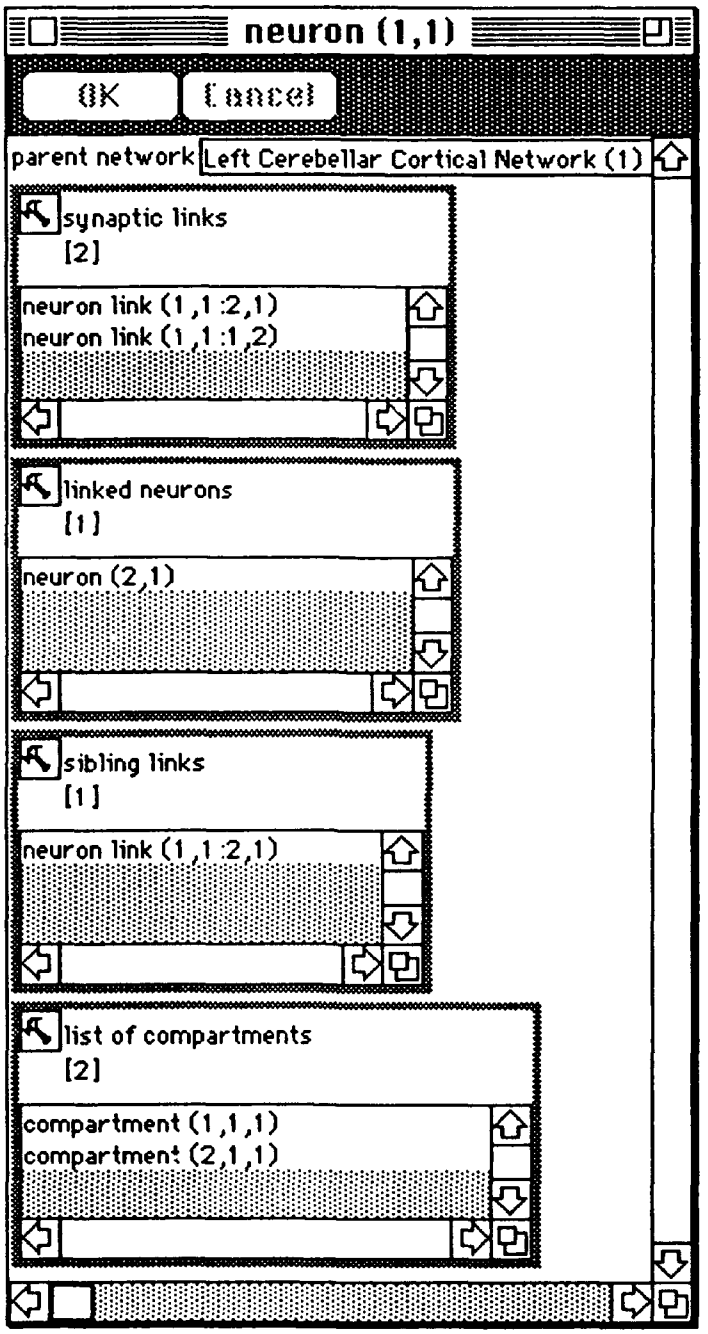
7.3. *Sibling Links* Sub-Window

This shows the list of sibling networks that it is connected to. Its sibling link is *Network Synapse (1:2)*.

7.4. *List of Neurons* Sub-Window

This shows the list of neurons that it is connected to. It has two neurons: *Neuron (1,1)* and *Neuron (2,1)*.

8. *Neuron Object* Window



8.1. *Synaptic Links* Sub-Window

This shows the list of neurons to which the current neuron is connected. The numeric indices in parentheses refer to the unique neuron and network identification numbers. It shows that *neuron (1)* in the *Left Cerebellar Network (1)* is

connected to the *neuron (2)* in the *Left Cerebellar Network (1)* as specified by *Neuron Link (1,1:2:1)*.

8.2. *Linked Neurons* Sub-Window

This shows the list of connected neurons. It is connected to *Neuron (2)*.

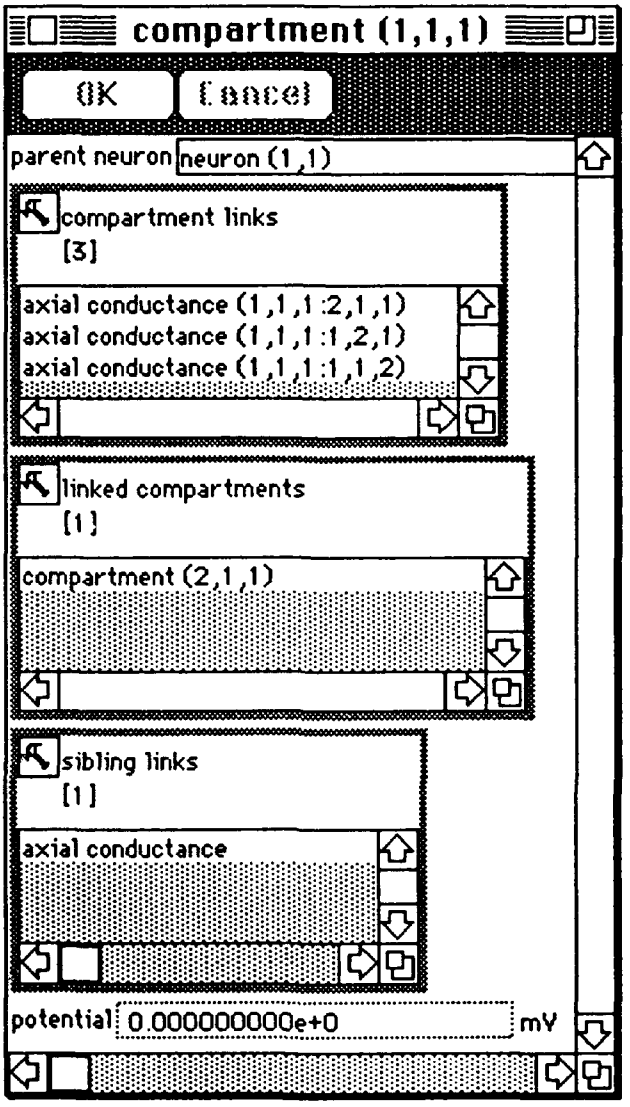
8.3. *Sibling Links* Sub-Window

This shows the list of sibling neuron to which it is connected. Its sibling link is *Neuron Link (1,1:2,1)*.

8.4. *List of Compartments* Sub-Window

This shows the list of compartments to which it is connected. It has two neurons: *Compartment (1,1,1)* and *Compartment (2,1,1)*. That is, *Compartment (1)* in *Neuron (1)* in *Left Cerebellar Network (1)* and *Compartment (2)* in *Neuron (1)* in *Left Cerebellar Network (1)*.

9. *Compartment Object* Window



9.1. *Compartment Links* Sub-Window

This shows the list of axial conductances to which the current compartment is connected. The numeric indices in parentheses refer to the axial conductance by their number (i.e., their name for identity).

9.2. *Linked Compartments* Sub-Window

This shows the list of connected compartments. It is connected to the *Compartments (2)* in *Neuron (1)* in *Left Cerebellar Network (1)*.

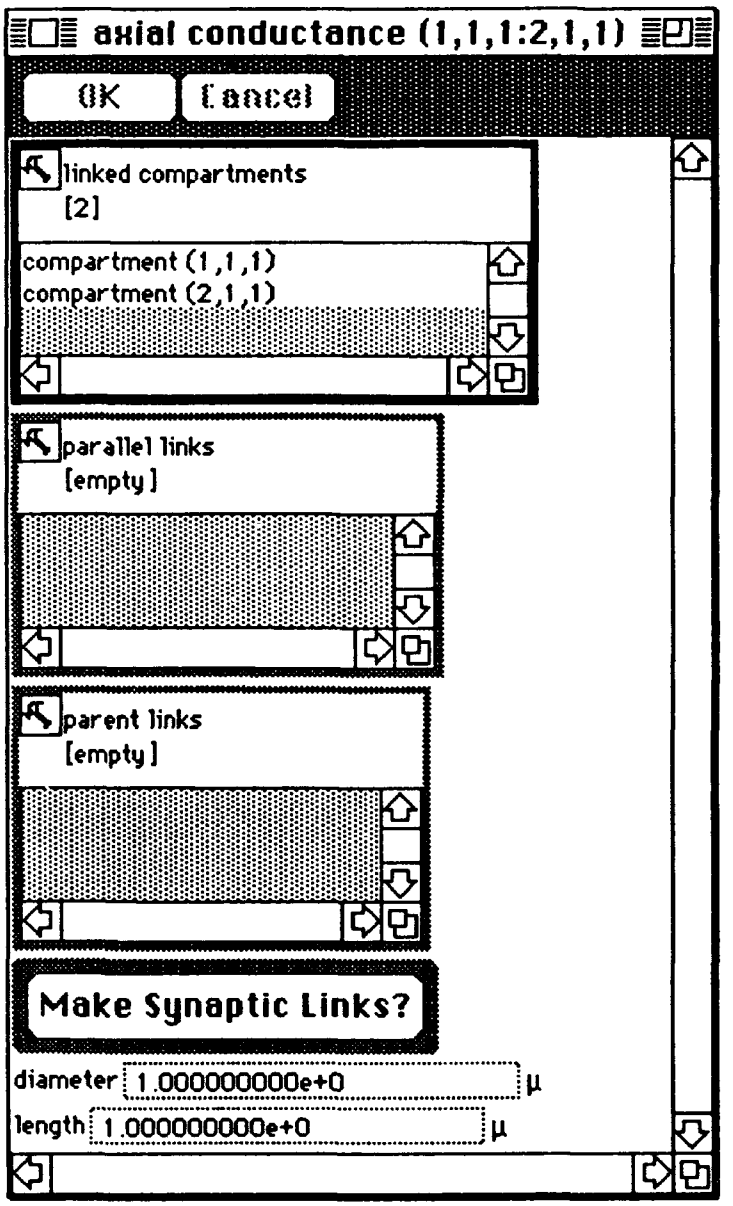
9.3. *Sibling Links* Sub-Window

This shows the list of sibling axial conductances to which it is connected.

9.4. *Potential* Box

This shows *potential* of the compartment in mV.

10. *Axial Conductance Object* Window



Axial conductance object includes the conductance, as well as the connection between adjacent patches of compartmental membranes.

10.1. *Linked Compartments* Sub-Window

This shows the list of connected compartments.

10.2. *Parallel Links* Sub-Window

This shows the list of sibling axial conductance to which it is connected.

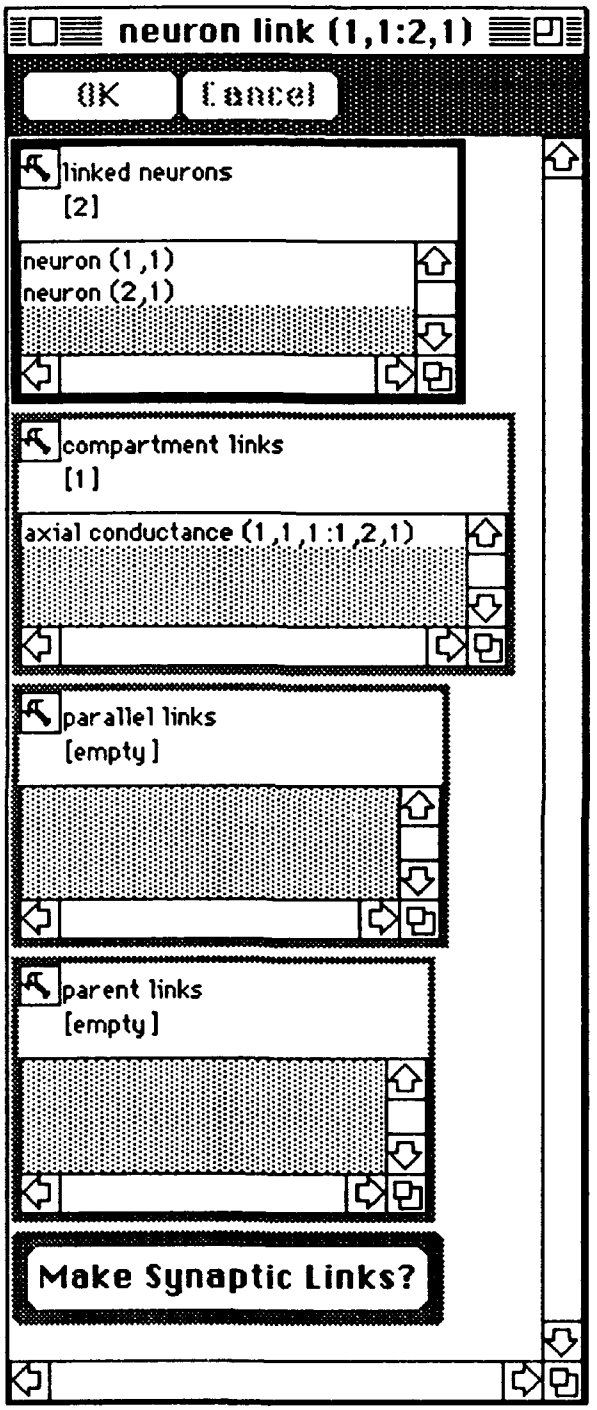
10.3. *Parent Links* Sub-Window

This shows the list of parent axial conductance to which the current compartment is connected.

10.4. *Make Synaptic Links* Button

It asks if synaptic links will be added.

11. *Neuron Link Object* Window



Neuron link object is the connection between adjacent neurons.

11.1. *Linked Neurons* Sub-Window

This shows the list of connected neurons.

11.2. *Compartment Links* Sub-Window

This shows the list of compartments to which it is connected.

11.3. *Parallel Links* Sub-Window

This shows the list of parallel links to which the current neuron is connected.

11.4. *Parent Links* Sub-Window

This shows the list of parent links to which the current neuron is connected.

11.5. *Make Synaptic Links* Button

It asks if synaptic links will be added.

12. Moving Objects to and from Other Objects

Parameter values and objects can be moved to and from other objects easily. All it needs is to hold down the `option`-key in the keyboard while pressing the mouse key over the selected item. An arrow icon () will show up with an outline of the selected item showing. That selected item can be "dragged", i.e., you can hold down the mouse and move it to the window of another object. "Drop in" the selected item into the appropriate window or an editable box will make the corresponding changes. If a parameter value is dragged and dropped into another parameter box, that value will be become the new value. If an object is dragged and dropped into another object link, then these objects will become connected.

13. Neuronal Simulation Language Description Macro

The neuronal simulation language can be used to describe the morphology (and parameters) of the neuron and neural network, as well as describe the flow of control of the run-time environment of the simulation. Special *macros* are groups of commands specified by the user to execute the described commands and procedures for the execution and construction of the neural simulation. The *macros* are specified in the text-editor window, and the corresponding *macros* commands appear as menu-items appended to the *Macros* Menu. Thus, the *macros* can be executed by the user easily with a single menu command.

13. Neuronal Simulation Language

The *macros* language can be written in the text-editor (the lower partition of the main window). It can be saved into a file for future references.

The language is very similar to the existing Pascal programming language with most of the Pascal syntax for creating variables, and for control such as "*for-loops*", "*repeat-loops*" and "*while-loops*". Data types of *integers, reals* and *booleans* are defined as in Pascal. Subroutines that help modularize groups of commands are also available in the language, and are implemented as *procedures* and *functions*. Such functions and procedures can be used as "libraries" for building neurons from its components, and for specifying similar structures by calling these procedures repetitively with one change in the parameter-list, so that similar neurons can be built with different parameters specified by a procedure call.

In addition to the standard Pascal facility, the *macros* are constructed like a procedure as shown in the following example:

```
macro make_neuron_macro_command "make_10_neuron_macro";
var
   cerebellar_neuron: object;
   i: integer;
begin
   for i:= 1 to 10 do
   begin
      cerebellar_neuron:= new(neuron);     {create 10 neurons}
      cerebellar_neuron.name:= "Cerebellar Neuron";
   end;
end;

macro make_brain_macro_command "make_rat_brain_macro";
var
   rat_brain: object;
begin
   rat_brain:= new(brain);      {create a brain}
   rat_brain.name:= "Rat Brain";
end;

macro make_networ_macro_command "make_network_macro";
var
   left_cerebellar_network: object;
begin
   left_cerebellar_network:= new(brain);  {create a network}
   left_cerebellar_network.name:= "Left Cerebellar Network";
end;
```

In the above example the first macro creates 10 cerebellar neurons, which will appear when the "*Check Syntax*" menu-item is executed. The *macro* is executed by pulling down the *Macro* Menu and selecting the "*make_10_neuron_macro*" menu-item.

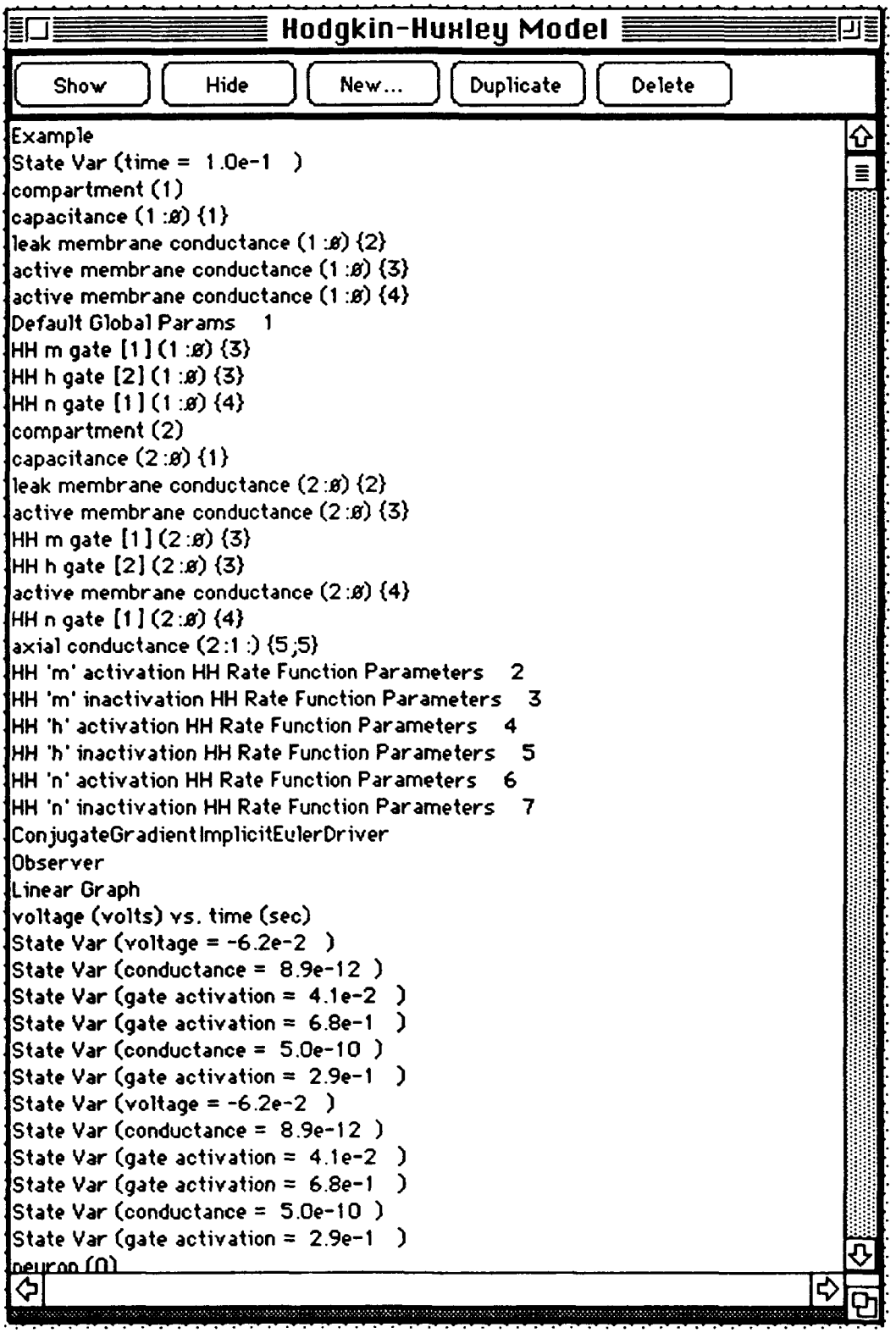
When the "*make_rat_brain_macro*" is executed, the second macro creates a brain and calls it "Rat Brain".

When the "*make_network_macro*" is executed, the third macro creates a network and calls it "Left Cerebellar Network",

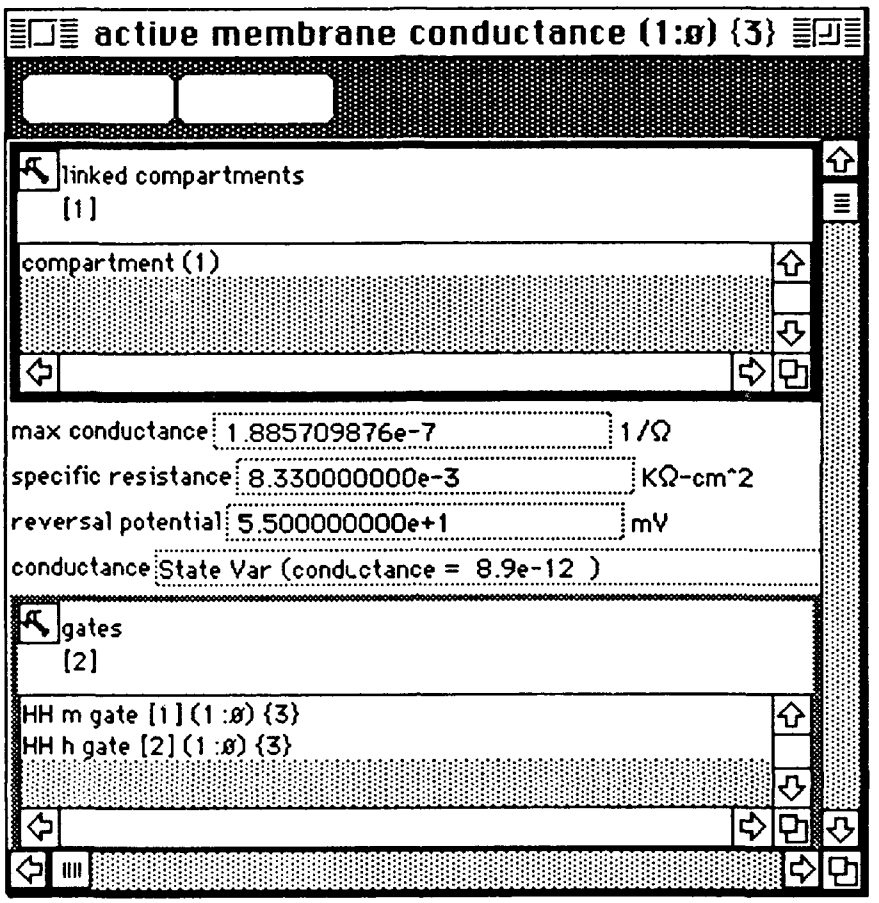
Appendix to User's Guide

14. Example 2

The file*Hodgkin-Huxley Model* simulates a compartmental neuron with voltage-activated conductances. A list of all the objects in the simulation appears below.



15. *Active Conductance* Window



Active Conductance includes a several new features that allow the user to simulated voltage gated activity.

15.1. *Linked Compartments* Sub-Window

Like the passive conductance windows described earlier, active conductances contain a sub-window which lists the compartments that it joins. In this case, there is only one compartment, and therefore the conductance is to ground by default.

15.2. *Max Conductance and Specific Resistance* Sub-Windows

The *max conductance* sub-window displays the maximum value of the conductance. This value is not directly editable but is computed from the *specific membrane resistance.* and the surface area adjacent membrane.

15.3. *Reversal Potential* Sub-Window

The reversal potential gives the potential difference across the conductance for which no current will flow.

15.4. *Conductance State Var* Sub-Window

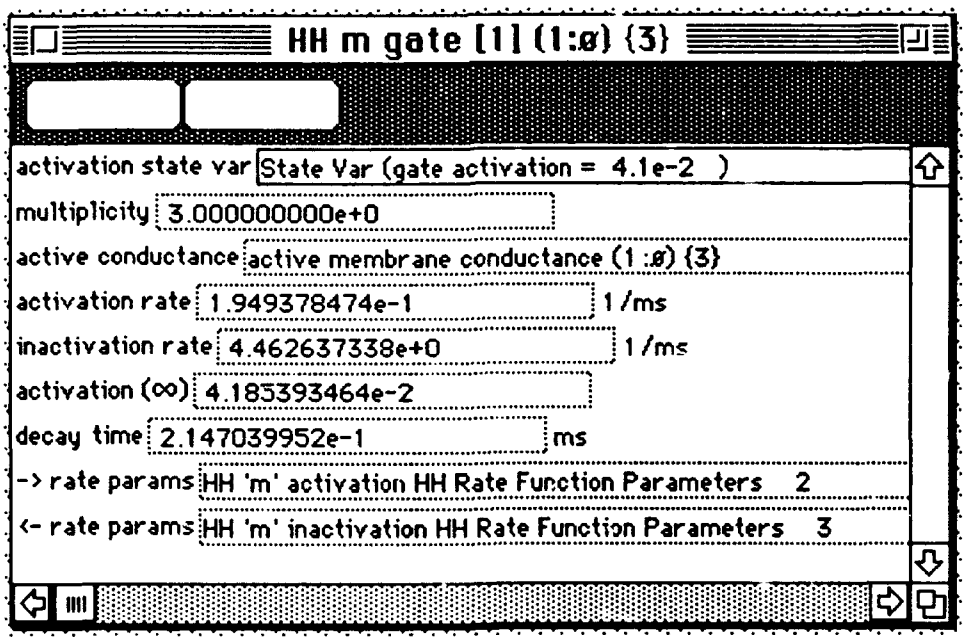
The *conductance state var* is a dynamical variable which records the current value of the conductance.

15.5. *Gate List* Sub-Window

The *Gate List* implements the active properties of *conductance*. They are described below.

16. *HH Gate* Window

The *HH Gate Window* describes a voltage activated gate. They are currently implemented in three varieties, *m*, *h*, and *n*, following the convention of Hodgkin and Huxley.



16.1. *Activation State Var* Sub-Window

The *activation state var* is a dynamical variable which records the current value of the gate activation.

16.2.*Multiplicity* Sub-Window

The *multiplicity* sub-window specifies the power or weight of the gate in determining the total conductance through the channel. It is typically an integer and may be thought of as the number of identical (and independent) voltage activated gates in series.

16.3.*Active Conductance* Sub-Window

Reference to the *active conductance* object.

16.4.*Activation Rates* Sub-Windows

Gate activation is governed by an equation of the form:

$$dm/dt = \alpha(1-m) - \beta m$$

where the activation (inactivation) rate is given by α (β). These are also displayed in terms of the steady-state gate activation and the inverse instantaneous time constant.

16.5.*Activation Rate Function Parameters* Sub-Windows

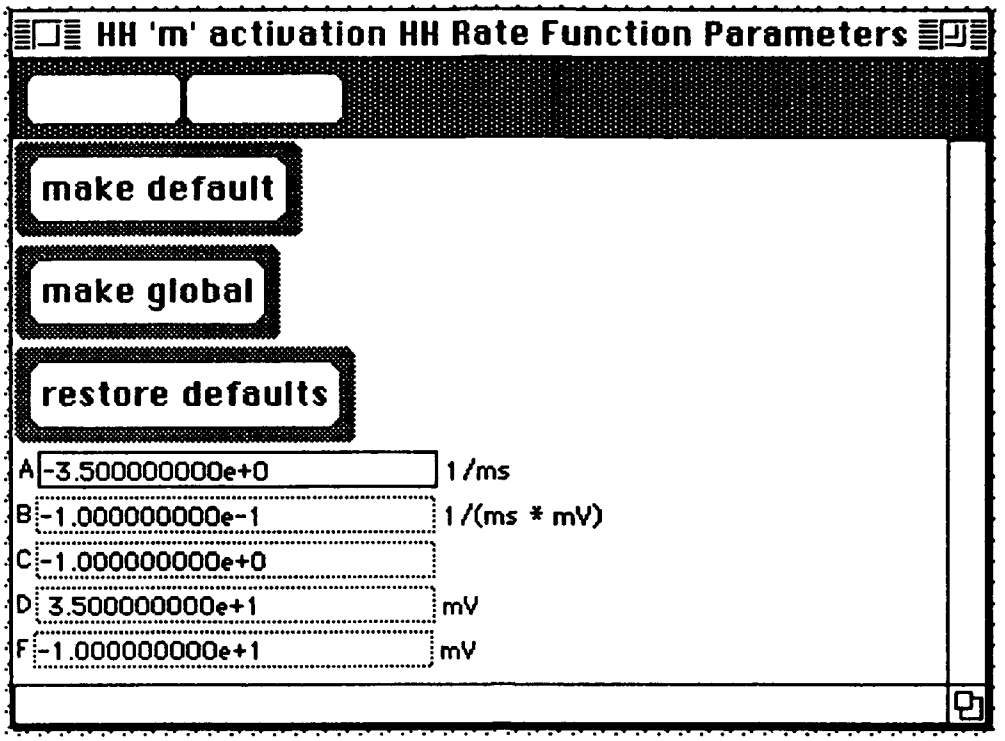
The instantaneous activation rate, α, is governed by an equation of the form:

$$\alpha(V) = (A + BV)/(C + exp[(V+D)/F]$$

where the parameters *A-F* are stored in separate parameters objects.

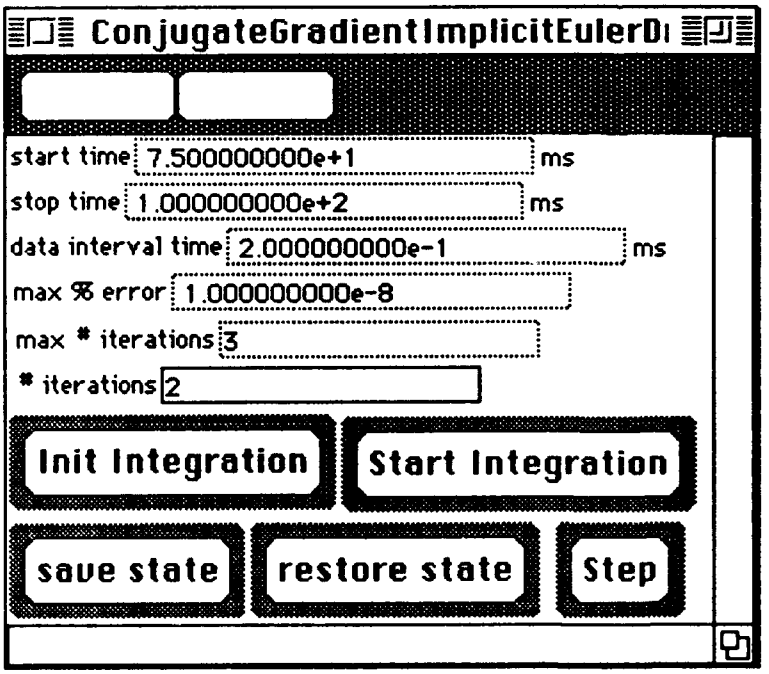
17. *HH Rate Function Parameters* Window

Gates in the model can have their own local parameters which can be shared by other gates by clicking on the *"make default"* button. This automatically causes all gates of the right type to use these parameters by default.



18. *Integration Driver* Window

The *integration driver* advances the simulation through time.



18.1 *stop, stop, and interval time* Sub-Windows

The *start, stop,* and *interval time* sub-windows control the duration and time resolution of the simulation.

18.2 *max % error, max # iterations, and # iterations* Sub-Windows

The *max % error* sub-window controls the number of conjugate gradient iterations per integration time step. Such iterations are performed until either the largest % change from the previous iteration is less than *max % error* or *max number iterations* has been exceeded. The *# iterations* window gives the actual number of iterations performed for that time step.

18.3 *Init and Start Integration* Command-Buttons

The *Init Integration* button initialized the integration. The integration must be initialize before the start button is clicked or an error message results. The *Start Integration* button begins the integration, which proceeds until the exit conditions are satisfied.

18.4 *Save and Restore State* Command-Buttons

The *Save* and *Restore* buttons save and restore the current or saved values of all dynamical variables.

18.5 *Step* Command-Button

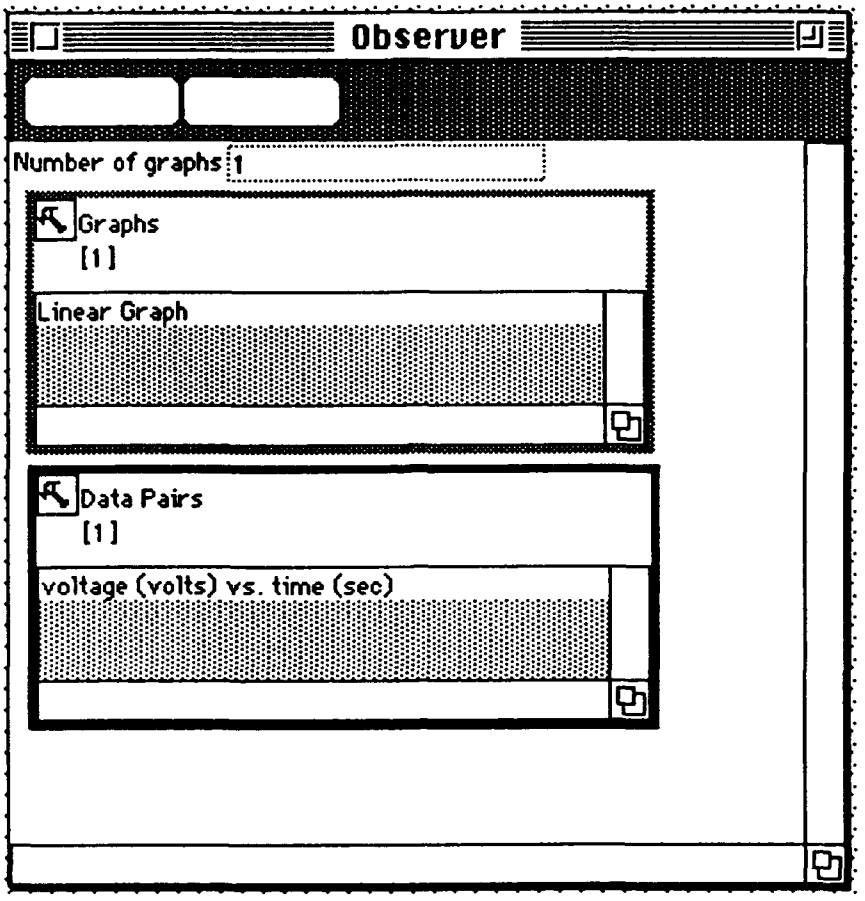
***Step* advances the simulation one time step.**

19. *Graphical Display*

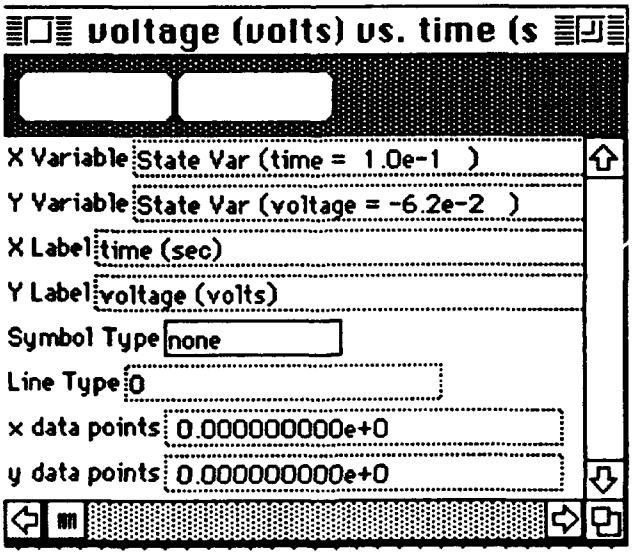
The current version of the application has been enhanced with graphical capabilities as the following example shows.

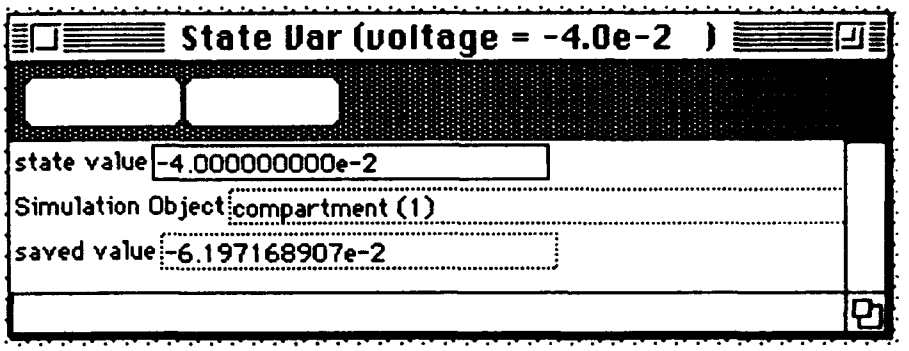
19.1 *Observer* Window

Events in the application can be observed through an "Observer" window, which allows the user to organize his "Graphs" and "Data pairs". Currently, the observer can only be driven by updates to the global time, but this is expected to change in future implementations.



19.2 *Data Pairs* and *State Vars*

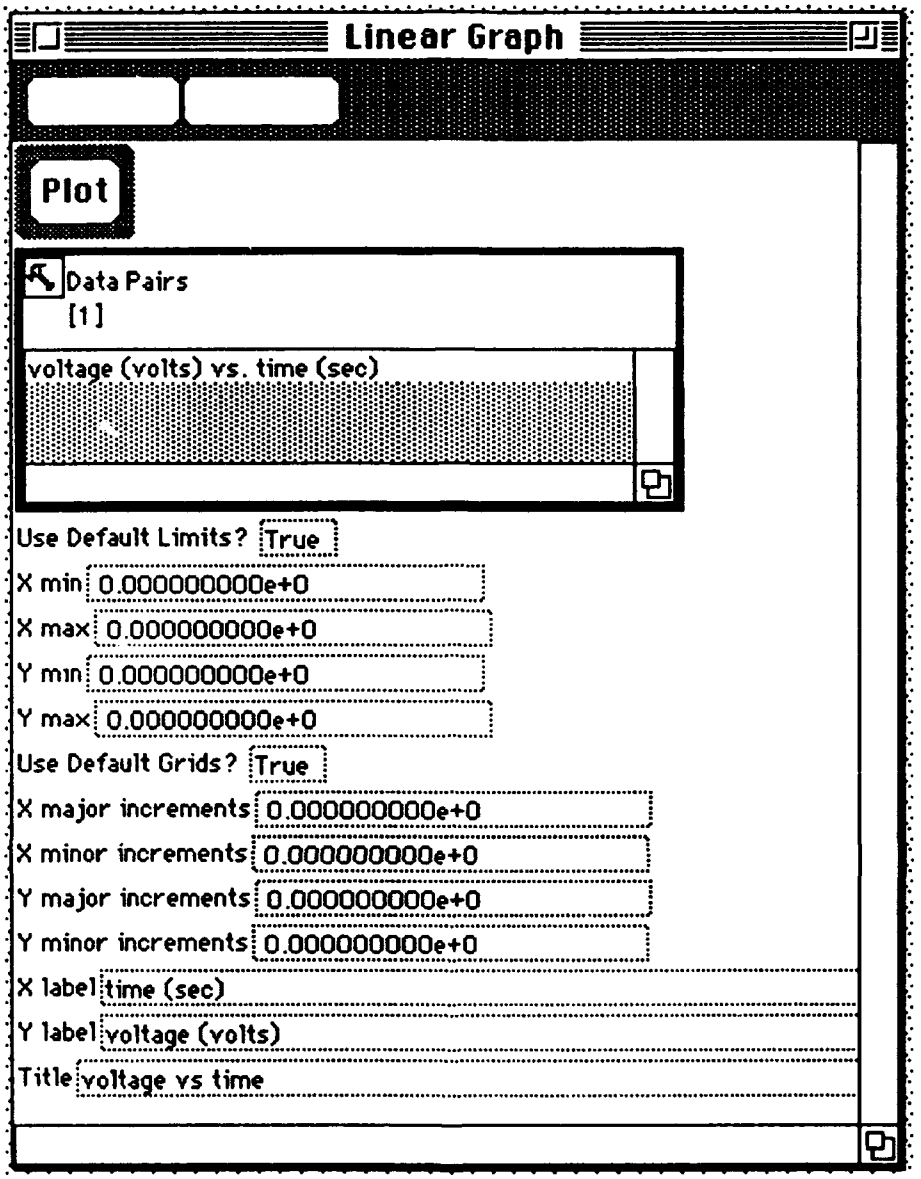




Dynamical variables (*State Vars*) are added to *Data Pairs* to construct coordinate pairs that can then be graphed. As show above, dynamical variables may also be edited directly by the user. Here, the membrane voltage has been forcibly depolarized by approximately 20.0 mV.

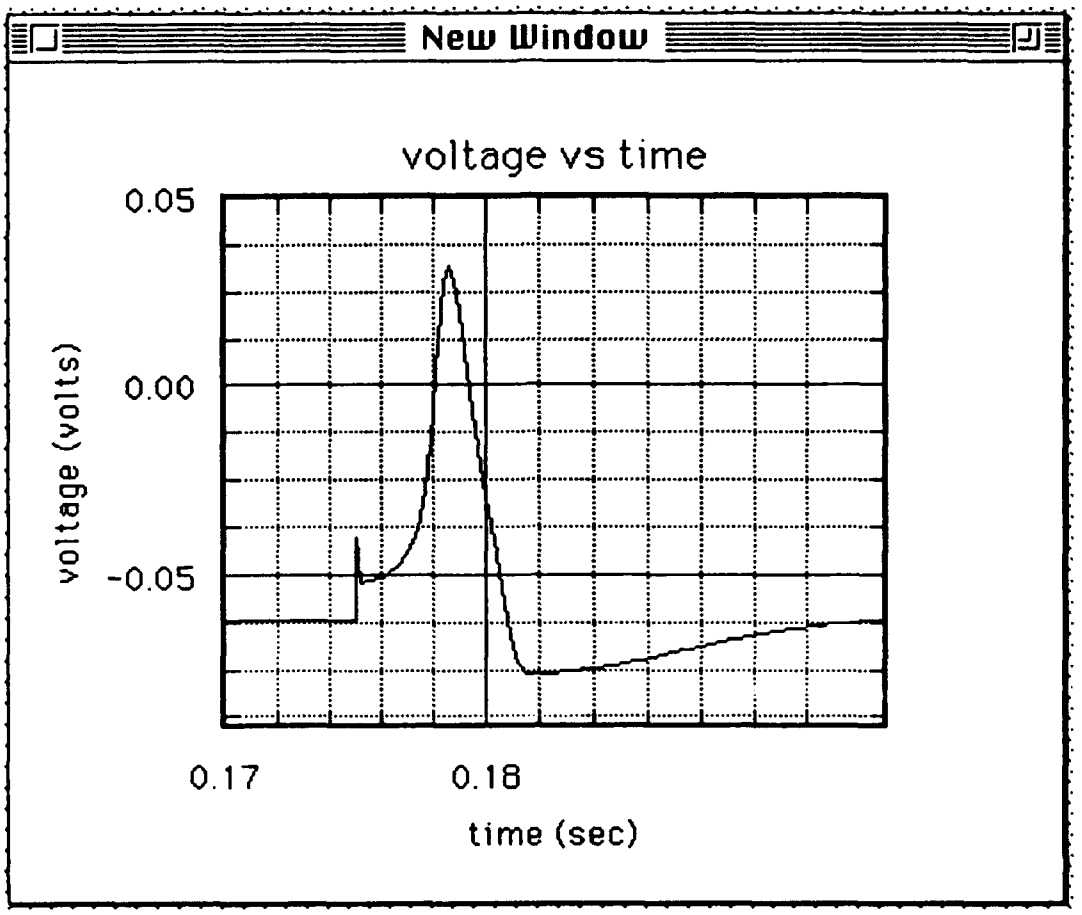
19.3 *Linear Graph* Window

Axis labels and title can be specified by the user by editing in the appropriate windows. Linear Graph uses default values for the other plot parameters unless overridden by the user.



19.4 *Plot* Window

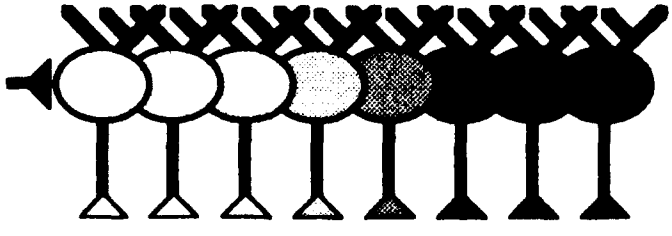
Clicking *Plot* in the *Linear Graph* produces a plot of the *data pair* record. This forced depolarization is evident from the plot.


New Window
voltage vs time
0.05
0.00
-0.05
voltage (volts)
0.17
0.18
time (sec)

MacNeuron

(version 0.3)

Tutorial



0 The MacNeuron Tutorial

0. 1 What is MacNeuron?

MacNeuron is a tool for simulating custom designed neural systems.

0. 2 About this Tutorial

This tutorial provides a self contained introduction to the *MacNeuron* application. Each example utilizes a step by step approach, allowing you to follow along on your own Macintosh computer. After completing this tutorial, you should feel confident enough to begin experimenting with *MacNeuron* on your own. Also consult the *MacNeuron Reference Manual* for a comprehensive review of all the features built in to the *MacNeuron* application.

note: *MacNeuron* is still very much in development and many features that will ultimately be part of the application have not yet been incorporated or are still getting the 'bugs' worked out . To avoid rewriting this tutorial with each revised version of *MacNeuron*, this tutorial has been written, as much as possible, with the 'final' version in mind. Occasionally, this creates discrepancies between what is written in the tutorial and what is actually appearing on the screen. Where such discrepancies occur, temporary text, such as this, may be inserted to clear up possible confusion. In such cases, the tutorial should be read as a guide to the final form that *MacNeuron* is expected to take.

1 Example I: Running the *MacNeuron* Application

1. 1 Getting Started

The *MacNeuron* icon (version 0.3) looks like this:

MacNeuron (v0.3)

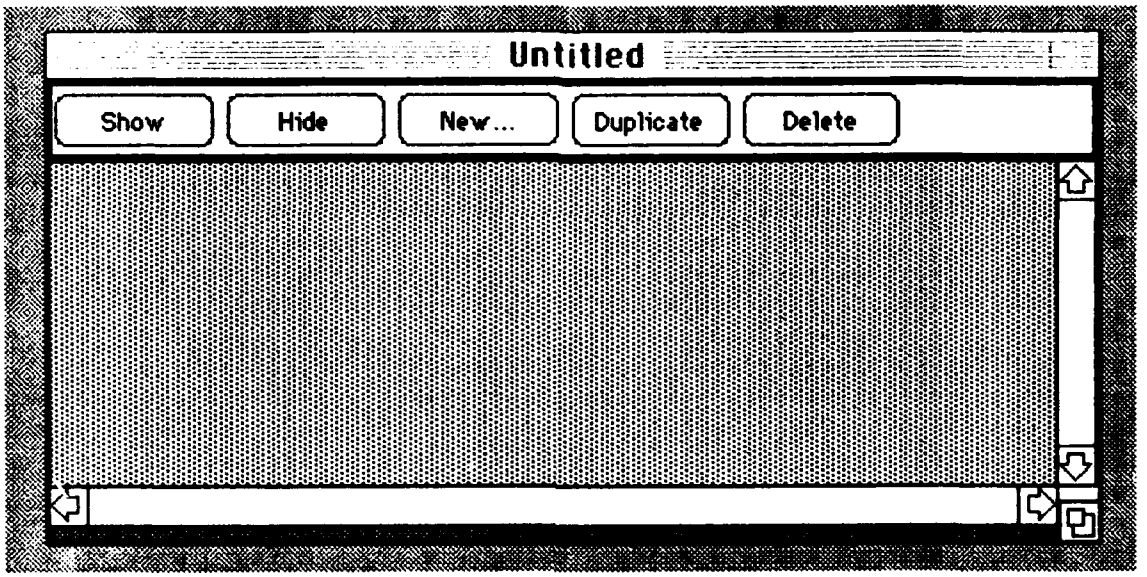
Click on the *MacNeuron* icon to select it:

MacNeuron (v0.3)

Start *MacNeuron* by double clicking on the *MacNeuron* icon. (Alternatively, you can select the *MacNeuron* icon and then choose "*Open...*" from the *File* menu).

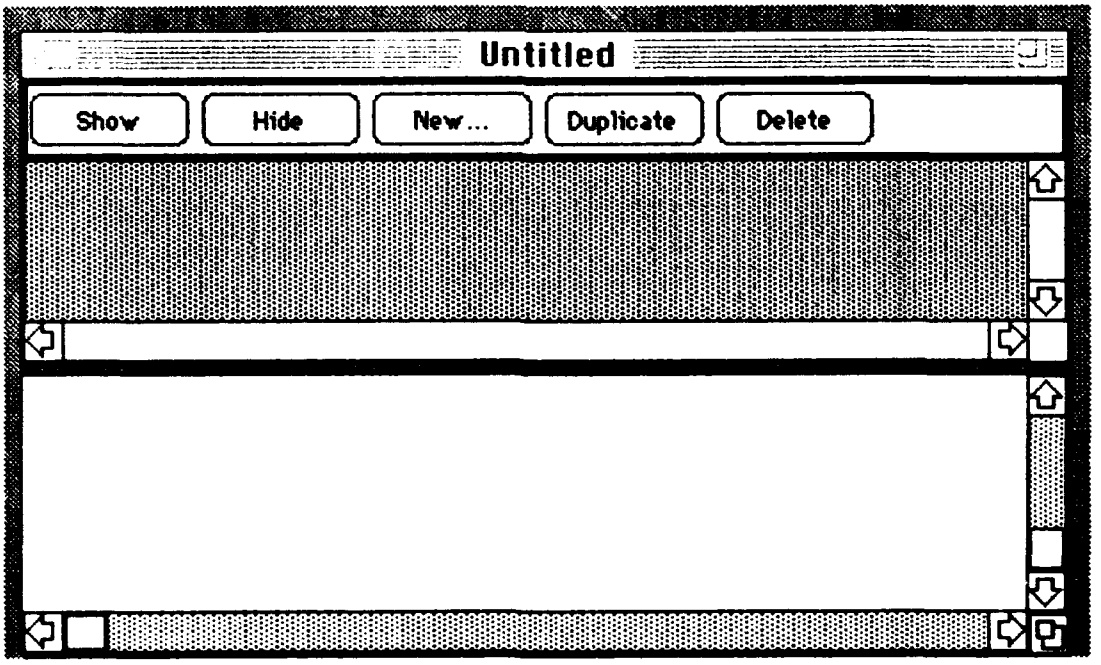
1.2 Main *MacNeuron* Window

MacNeuron will start up with a *New* window called *Untitled.* This is the *main window* of the application. The name of this main window will change when the file is saved, just like any other standard Macintosh program.



1.3 Resizing the Main *MacNeuron* Window Partitions

This main window contains two parts (or partitions). When the resize box at the lower right corner of the window is enlarged, the second half (lower partition) of the window will show up:



By holding down the mouse button at the window-partition divider (the shaded thick line between the two partitions) and "draging" the divider up or down, the relative partition sizes can be adjusted. Note that the cursor changes from a pointer to a cross-hair when it is on the window partition divider.

1.4 Simulation List (Upper Partition)

The upper partition of this main window consists of nested lists containing all the objects (neurons, conductances, compartments, networks, drivers, graphs, etc...) currently in the simulation. The upper portion of the window also provides a palette of tools for performing useful operations on the listed items. For instance, selecting a particular neuron from the list and then clicking the show button will cause the window for that neuron to be displayed. There are also tools for hiding, deleting and duplicating any of the objects in the list. Examples of these operations are provided later in the tutorial.

1.5 Simulation Language (Lower Partition)

The lower partition of the main window is a text window where circuits may be specified using a high level language, called "*macro*", which provides the same functionality as the iconic user interface, but offers a powerful alternative for performing large numbers of similar operations.

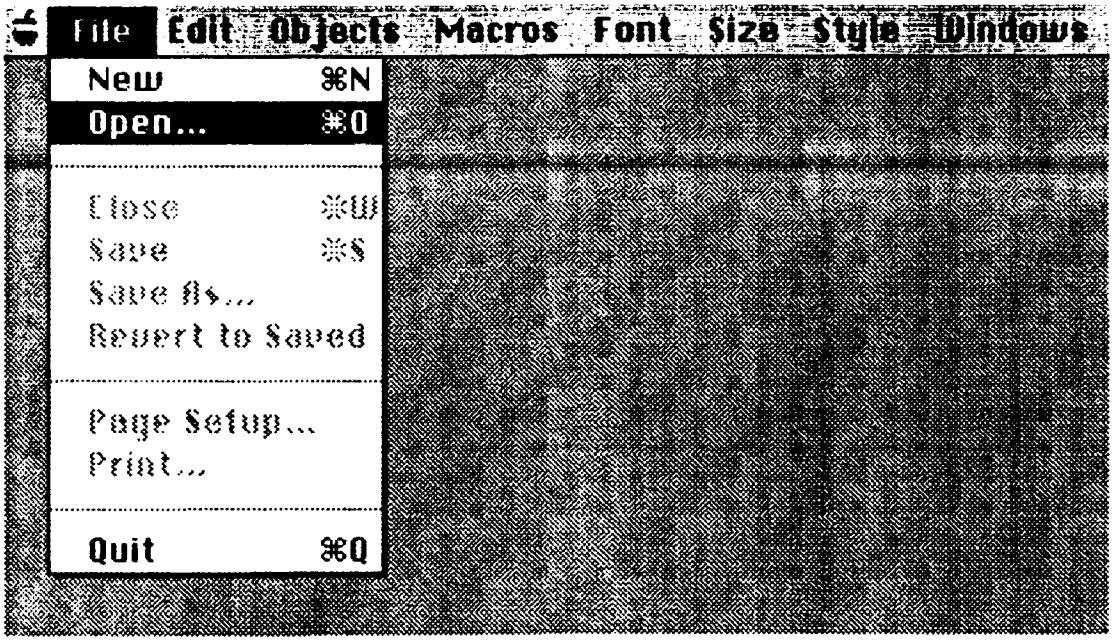
1.6 What 's Next?

We are now ready to go the next example. To close the *Untitled* window, click in the close box in the upper left corner. It is not necessary to close the *Untitled* window in order to proceed, but doing so will reduce unnecessary clutter on your screen.

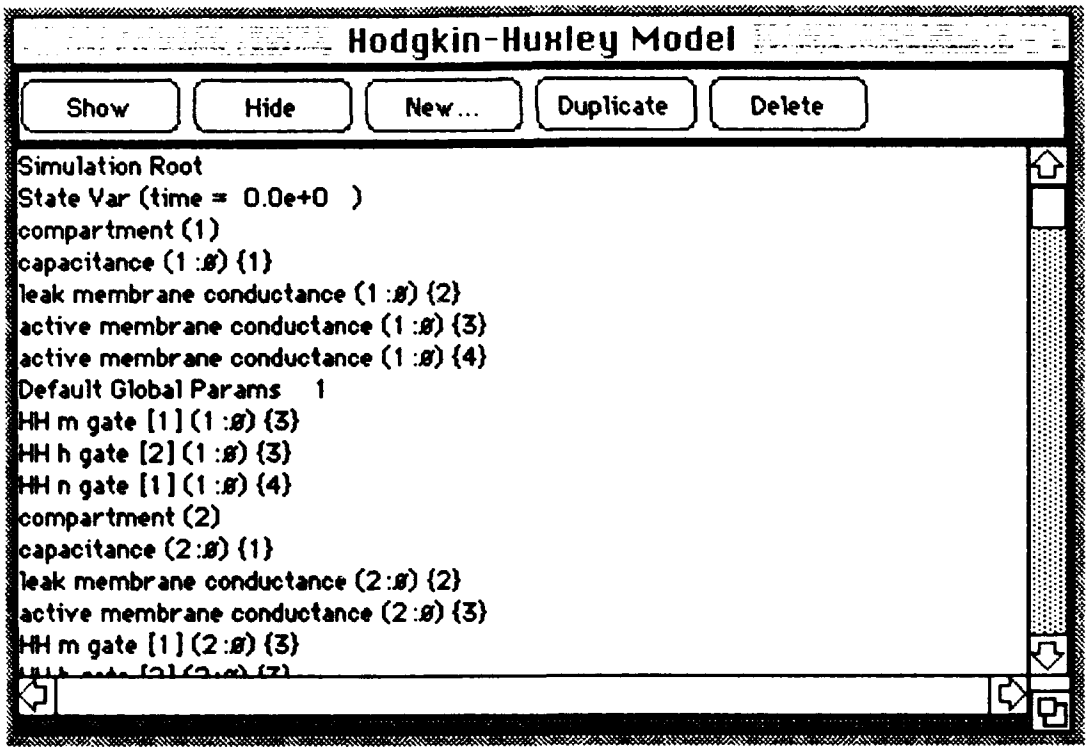
2 Example II: Simulating a Previously Constructed Neural Circuit

2.1 Opening Existing Files

At the top of the screen is the menu bar. To open an existing file, pull down the *File* menu from the menu bar and select "*Open...*"



A standard dialog box will appear asking you to select a file. Open the file called *Hodgkin-Huxley Model* (it would be a good idea copy this file first) :



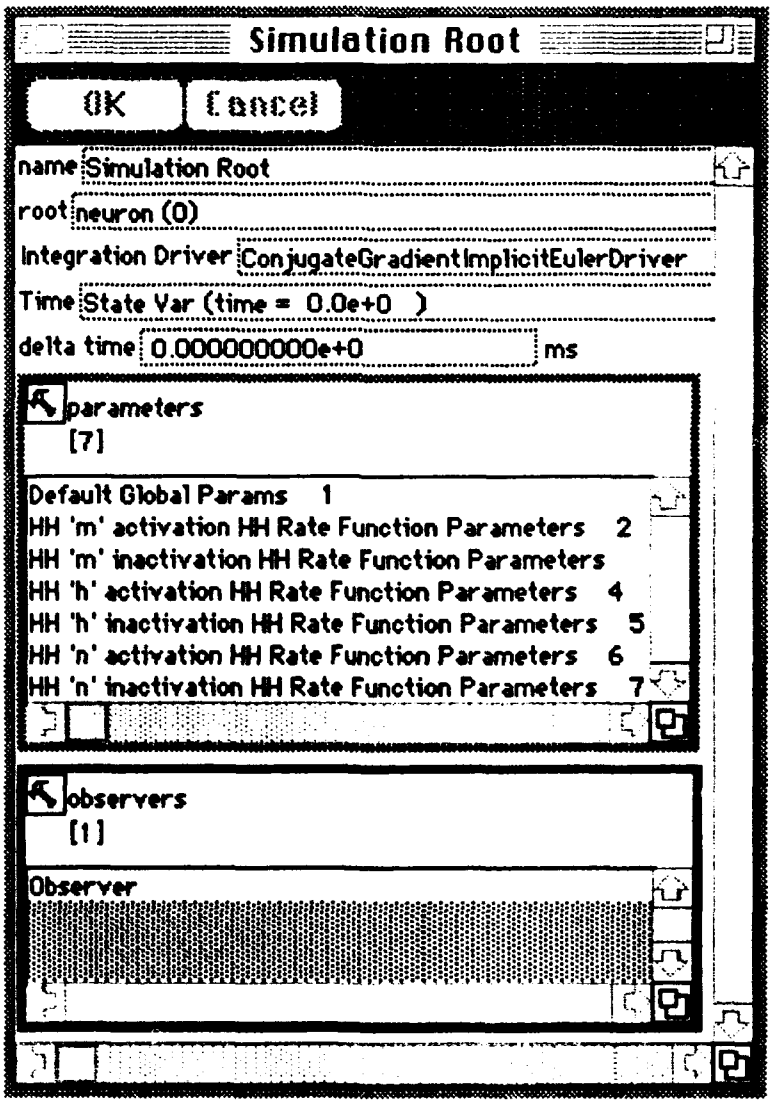
Hodgkin-Huxley Model is a two compartment circuit representing a short axonal segment containing active membrane conductances. The main window contains a single object called "Smulation Root'". As we will see, everything in the simulation "grows out" of the Simulation

Root, which therefore acts as a landmark from which any other object in the simulation can be found.

The simulation list currently contains all the objects in the simulation. Usually, the *Simulation Root* is the first object in the list.

2.2 *Simulation Root* Window

Double click on the *Simulation Root* to open its window (alternatively, select the *Simulation Root* by clicking on it once and then click "*Show*" in the main window's command pane):



You will have to manipulate the various panes inside the *simulation root* window in order to achieve the configuration shown. Currently, the default window organization is not ideal. This will be the case with essentially all the windows appearing in the tutorial.

The *simulation root* points to all the main object categories, or hierarchies, which make up the simulation:

root neuron (0) : The top of the physiological hierarchy. This could also be a network or a brain.

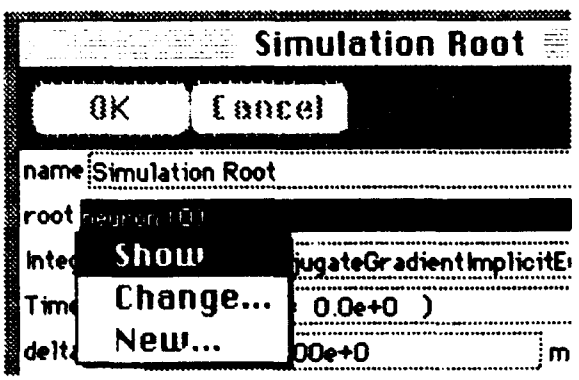
Integration Driver ConjugateGradientImplicitEulerDriver : The object which implements the particular numerical integration algorithm chosen to drive the simulation. The currently selected driver uses conjugate gradient descent to solve a matrix of coupled linear equations resulting from an implicit integration time step.

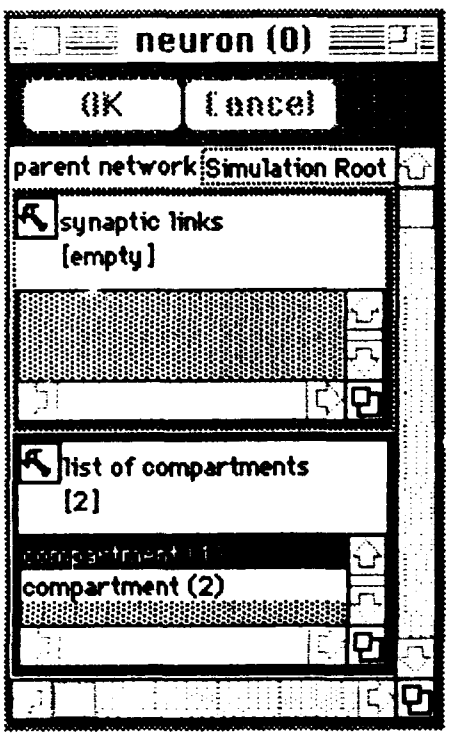
parameters : Objects in this list contain parameters which can be shared by multiple simulation objects. This allows the same parameters, such as those characterizing voltage dependent conductance channels, to be used by more than one voltage-activated gate.

observers : List of objects which contain other objects for recording and ploting simulation data.

2.2 *Root Neuron* Window

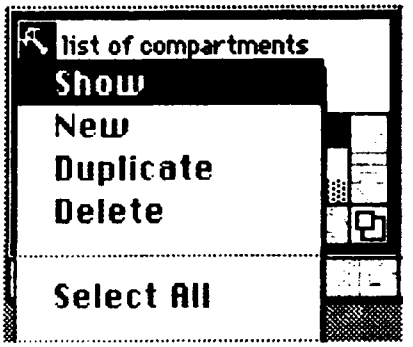
Select *Show* from the root neuron's pull down menu to activate its window:





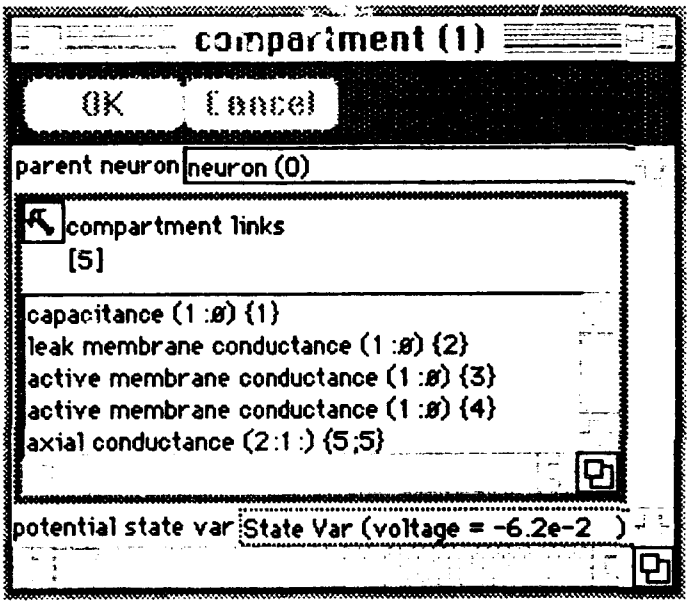
Neuron (0) contains two compartments and has no synaptic links. Since *neuron (0)* is the root physiological object, the network which contains it

(parent network) is the *simulation root*. Click on the text "*compartment (1)*" (in the list of compartments sub-window) to highlight it:

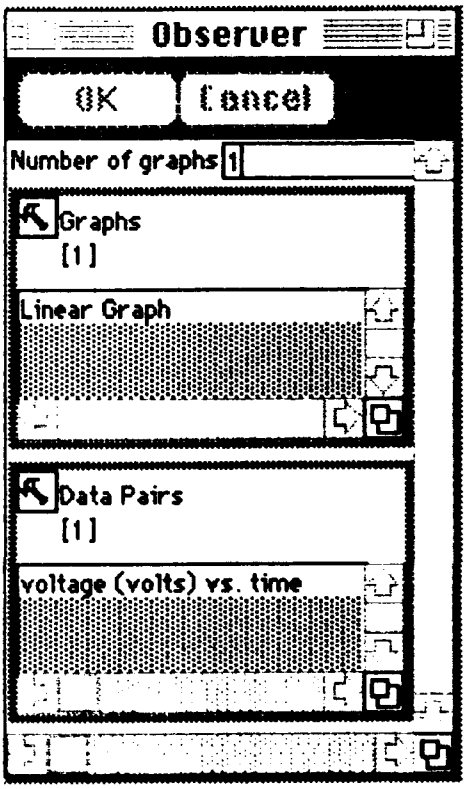


2.2 Compartment Window

Activate the window for *compartment (1)* by double clicking on its text or by selecting Show from the tools pull down menu (You may also wish to close the *neuron (0)* window as it will no longer be needed).



Compartment (1) displays its parent neuron, in this case the *root neuron*, *neuron (0)*, a list of links to other compartments (or to "ground" as indicated by a 'ø' symbol) and the electrical potential. Keep this window available since we will observe the potential in this compartment during the simulation:



2.3 Observers

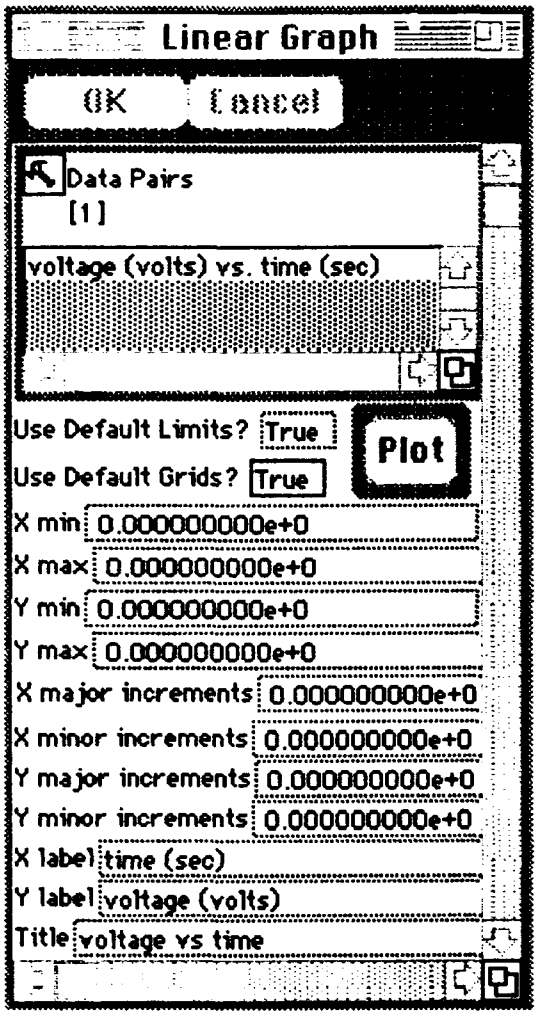
Return to the *Simulation Root* and activate the *Observer* located in the *Observers* sub-window:

The *Observer* watches the specified *Data Pairs* and ensures that new values are recorded as the simulation progresses. The *Observer* also keeps track of the different graphs employed to display the data.

2.4 Graphs

Many of the graphical capabilities ultimately to be included in *MacNeuron* are still being developed. What is described here represents an intermediate stage. Some of the windows employed represent temporary 'patches' and therefore possess minimal functionality. Such windows will be replaced in the next version of *MacNeuron*.

Activate *Linear Graph* found in the *Graph* sub-window (You may also want to close the *Observer* window as it will not be needed for the following).



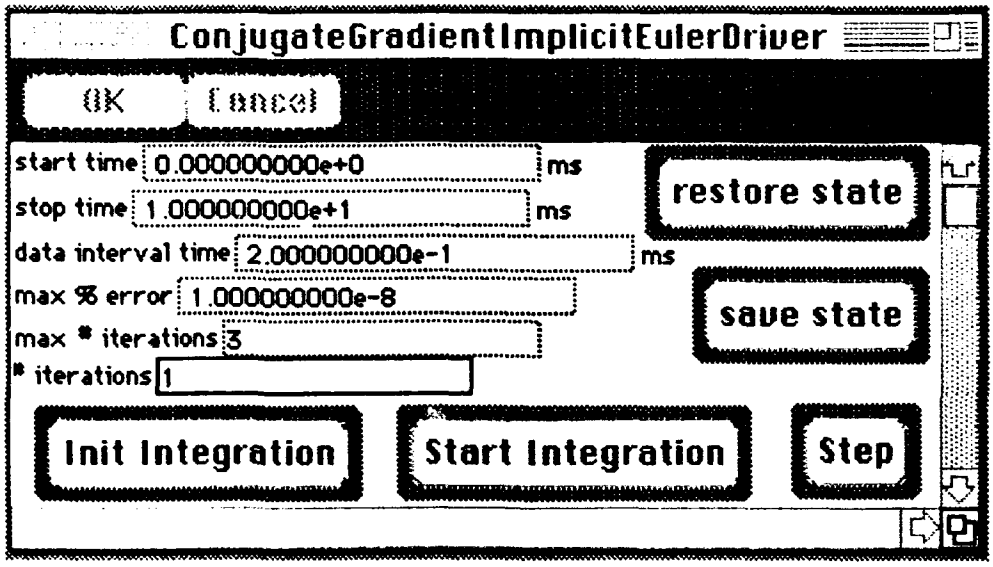
Linear Graph contains a list of those *Data Pairs* to be ploted, as well parameters for specifying the range of data to be included, intervals between major and minor tic marks, axis labels, etc... There is also an option for generating reasonable default values for these parameters.

Save this window as it will be needed to plot the results of the simulation.

2.5 Drivers

Return to the simulation root and activate the *Conjugate Gradient Implicit Euler Driver* located in the *integration driver* sub-window:

The *integration driver* contains sub-windows for specifying the end points of the integration, the length of each integration time step, the maximum percentage error (needs to be set very low due to the inherent stiffness of the circuit), the maximum number of iterations (number of conjugate-gradient descents) per time step and the actual number of iterations used during the previous time step.



There are also several command buttons. A particular state, such as the equilibrium state, can be saved and then later restored using the buttons *"save state"* and *"restore state"*. The *"init integration"* button initializes the integration (used only after everything in the simulation is in place) and the simulation may be performed either in step mode or all at once, using the *"step"* and *"start integration"* buttons, respectively.

2.6 Running

To establish a baseline, hit the *"init integration"* and *"start integration"* buttons in the integration driver window (*Hodgkin-Huxley Model* should have started out in steady state but you can hit the *"restore state"* button in the integration driver window to be sure). The cursor will change to a stop watch while the simulation is running.

At the moment, the cursor does not change shape while the simulation is running and nothing on your screen responds to mouse clicks during this time. Additionally, the integration algorithms have not yet been optimized and therefore integrations may take a little while, depending on which computer you are using (the above baseline calibration takes a few seconds on a IIci).

Activate the widow for *compartment (1)* (use the Window menu if it has become buried).

Alas, the window menu is yet correctly implemented, so you will have to move things out of the way in order to find what you're looking for.

Since we are at steady state, it is necessary to change the potential from the steady state value (in this case by hand) in order to get something going. Select the *potential* sub-window and change the voltage to -40.0 *mV*.

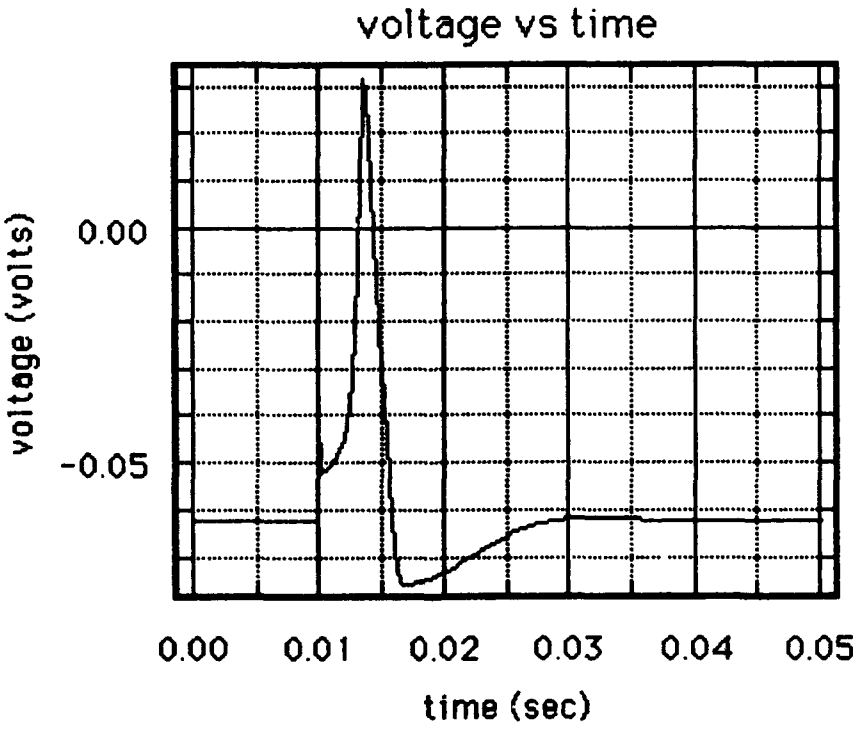
To facilitate development, *State Var* objects have been temporarily introduced into *MacNeuron*. Open a *State Var* just like any other window and edit its value in the usual way. A very important point to remember is that all *State Var* objects use MKS units, so 40.0 *mV* must be written as 0.040. *State Var* objects will be removed in the next version of *MacNeuron*.

You can now close of the *compartment (1)* widow.

Activate the integration driver window, change the start and stop times to 10.0 and 50.0 *msec*, respectively, and restart the integration.

2.7 Ploting

Activate the *Linear Graph* window and hit the *"Plot"* button. A plot of the voltage in *compartment (1)* vs *time* should appear:



Currently, *"Plot"* uses a temporary window with almost no functionality. One pathological feature of this window is that the grow box is hidden. Clicking where you expect the grow bow to be brings it into view. This window does

not close in the usual way either. as clicking in the close box does nothing. As mentioned previously, this window is merely a temporary patch while more sophisticated plotting windows are being developed.

2.8 What's Next?

Select *"Close"* from the *File* menu. *MacNeuron* will prompt you to save your changes, including the results of the simulation, and then all windows associated with *Hodgkin-Huxley Model* should disappear.

MacNeuon does not currently save simulation data, so there is no reason to save any of your changes to *Hodgkin-Huxley Model*. This will be fixed in the next version of *MacNeuron*.

Now that we have seen a bit of how *MacNeuron* works, we are ready to learn how to build neural circuits from scratch.

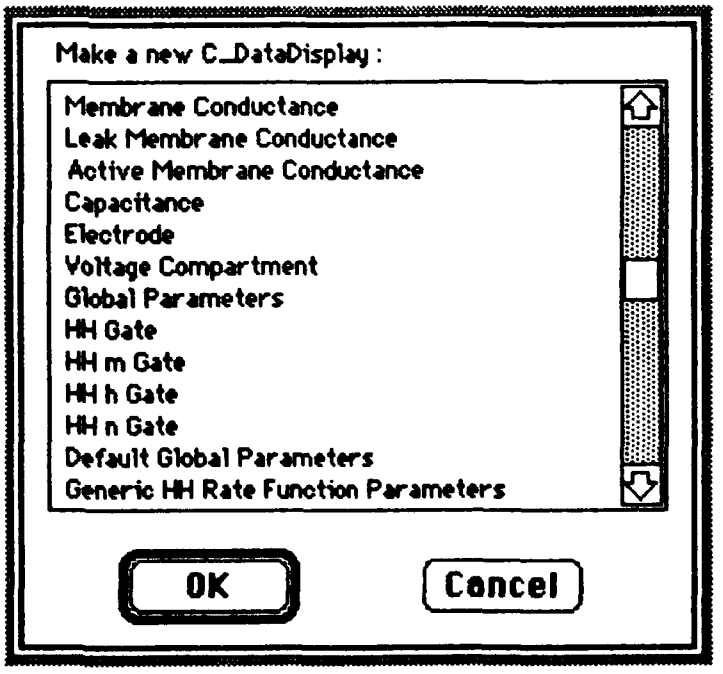
3 Example III: Building Neural Circuits

3.1 New Files

Select *"New..."* from the *File* menu in the *MacNeuron* menu bar at the top of the screen. A new main window, called *"Untitled"*, will appear on the screen (if you have just launched *MacNeuron*, or do so now, the *"Untitled"* window will come up automatically). Choose *Save* from the *File* menu and save this file as Example III.

3.2 Making a *Simulation Root*

The first object in any simulation is the *Simulation Root*. Click on *"New..."* in the main window's command pane. This causes the following dialog box to appear:



Only a *Simulation Root* can be made at this point, so this is the only object which appears.

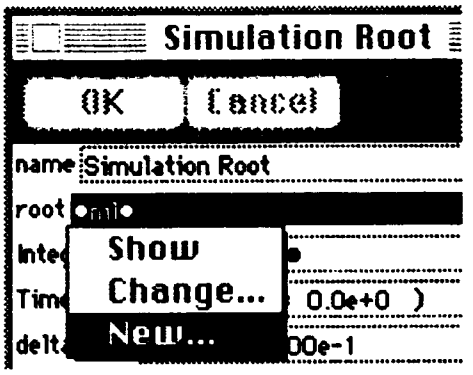
Currently, the user is presented a list containing all possible objects.

To make a *Simulation Root*, either select *"Simulation Root"* and hit *"OK"* (*"Return"* accomplishes the same thing) or double click on *"Simulation Root"*.

Simulation Root may be also called *Global Parameters*.

3.3 Making a *Root Neuron*

Select *"New..."* from the pull down menu in the *Root* sub-widow:

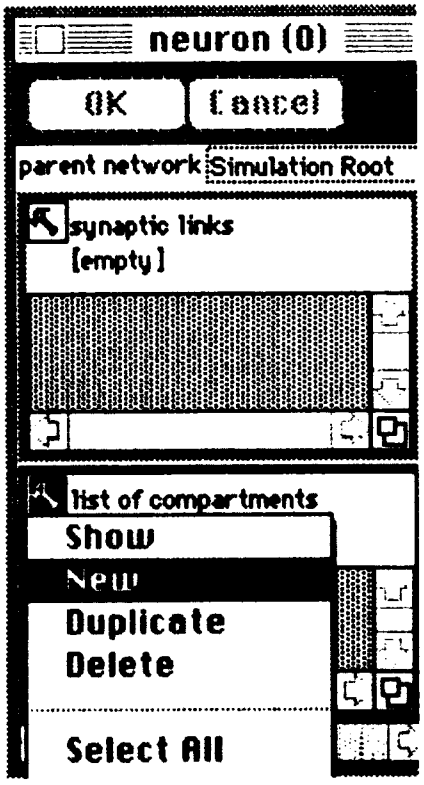


Select *"neuron "* from the *"Make a new"* dialog box and click *"OK"*.

3.4 Making a *Compartment*

Select *"Show"* from the pull down menu in the *Neuron Root* sub-widow containing *neuron (0)* to activate its.window.

Select *"New"* from the *"tools"* pull down menu in the list of compartments sub-widow (in the *neuron (0)*.window):



Select *"Voltage Compartment"* from the *"Make a new"* dialog box and click *"OK"*.

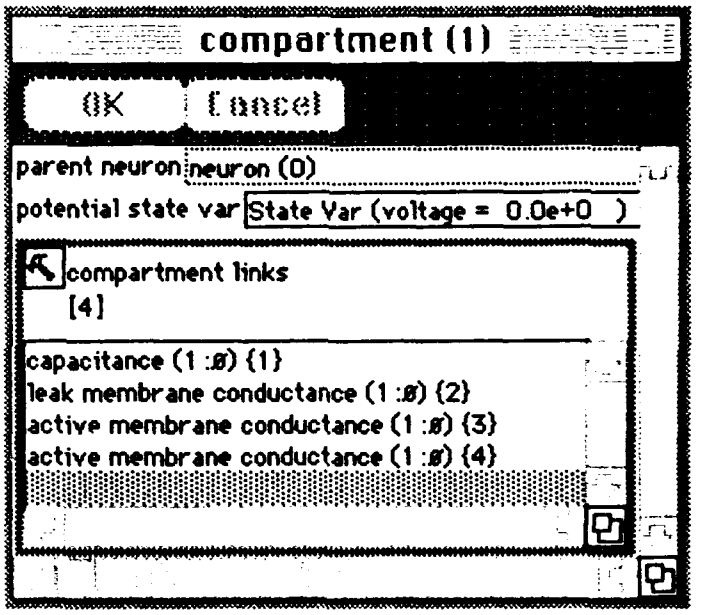
3.5 Making *Compartment Links*

In the *list of compartments* sub-widow, double click on the text *"compartment (1)"* or select *"Show"* from the *"tools"* pull down menu while *"compartment (1)"* is selected.

Select *"New"* from the *"tools"* pull down menu in the compartment links sub-widow (in the *compartment (1)*. window):

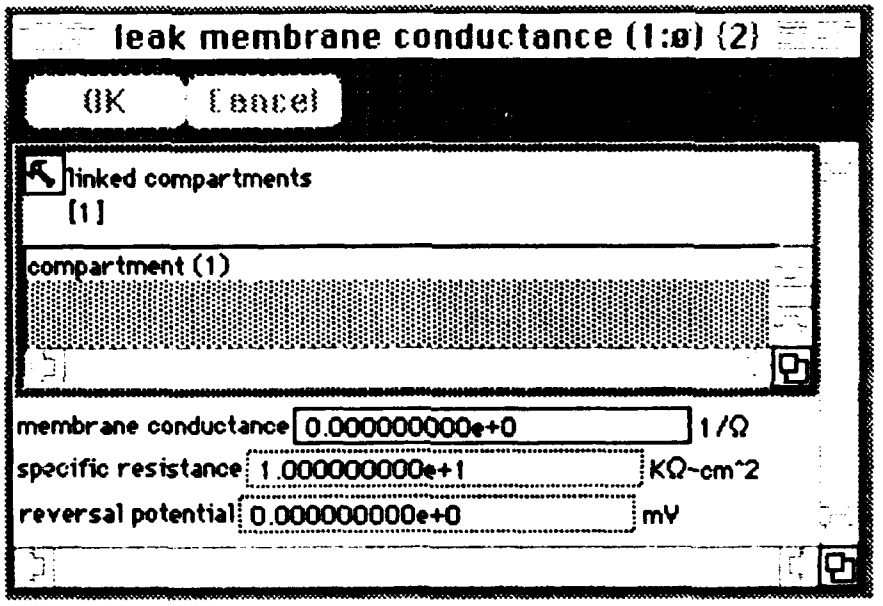
Select *"Capacitance"* from the *"Make a new"* dialog box and click *"OK"*.

Make a new *"Leak Membrane Conductance"* and two new *"Active Membrane Conductances"* in the same way. There should now be four items in the *compartment links* sub-widow.



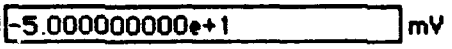
3.6 Setting the *Leak Membrane Conductance*

Double click on the text *"Leak Membrane Conductance (1:ø) {2}"* in the *compartment links sub-widow* (in the *compartment (1)*. window):



Change the *specific resistance* to 3.0 *kΩ-cm^2* and click *"OK"*:

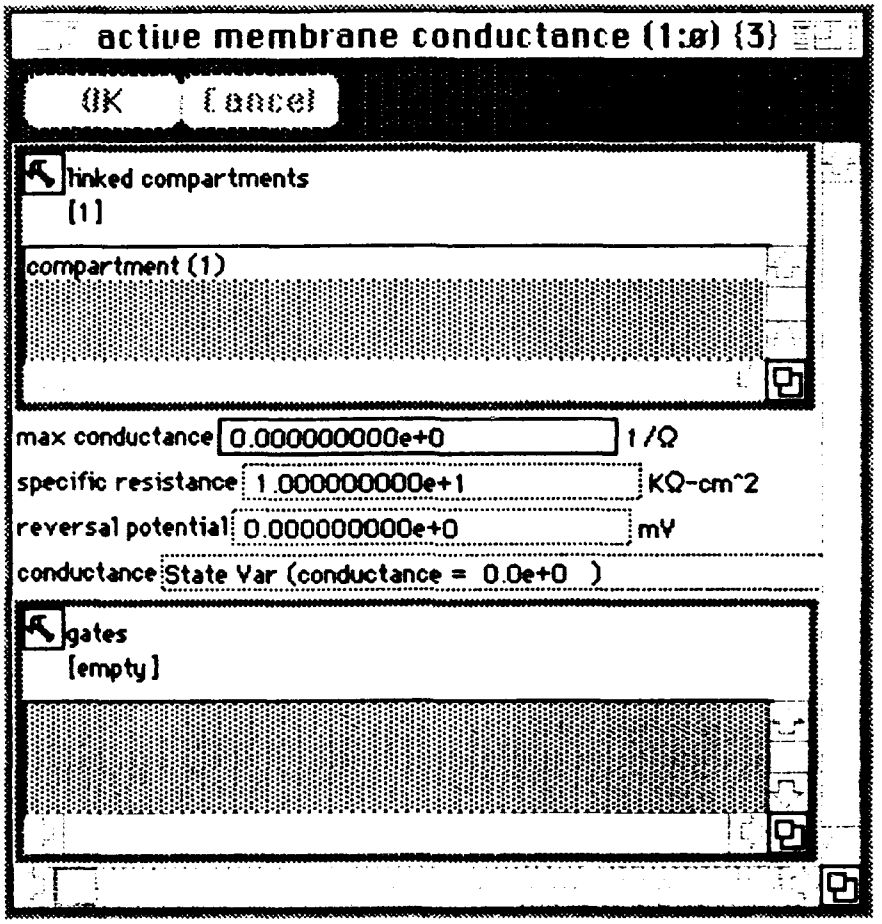
Change the *reversal potential* to -50.0 *mV* and click *"OK"*:



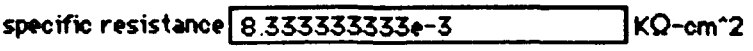
Close the *Leak Membrane Conductance* window.

3.7 Setting the *Active Membrane Conductance:* Na^+ channel

Double click on the text *"Active Membrane Conductance (1:ø) {3}"* in the *compartment links* sub-widow (in the *compartment (1)*.window):



Change the *specific resistance* to 0.0083... *kΩ-cm^2* and click *"OK"*:



Change the *reversal potential* to 55.0 *mV* and click *"OK"*:

Change the *conductance* to 8.7e-12 *1/Ω* and click *"OK"*:

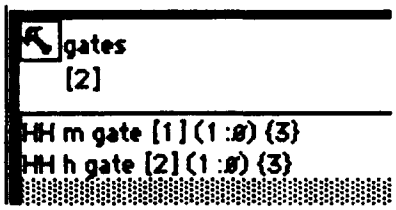
Select *"New"* from the *"tools"* pull down menu in the *gates* sub-widow:

Select *"HH m Gate"* from the *"Make a new"* dialog box and click *"OK"*.

Make a new *"HH h Gate"* as well. There should now be two items in the *gates* sub-widow.

3.8 Setting the *HH m Gate*

Double click on the text *"HH m Gate [1] (1:ø) {3}"* in the gates sub-widow (in the *Active Membrane Conductance (1:ø) {3}* window):



Gate activation is governed by an equation of the form:

$$dm/dt = a(1-m) - bm$$

where the activation (inactivation) rate is given by α (β). These are also displayed in terms of the steady-state gate activation and the inverse instantaneous time constant.

Select *"New"* from the *"-> rate params"* sub-window:

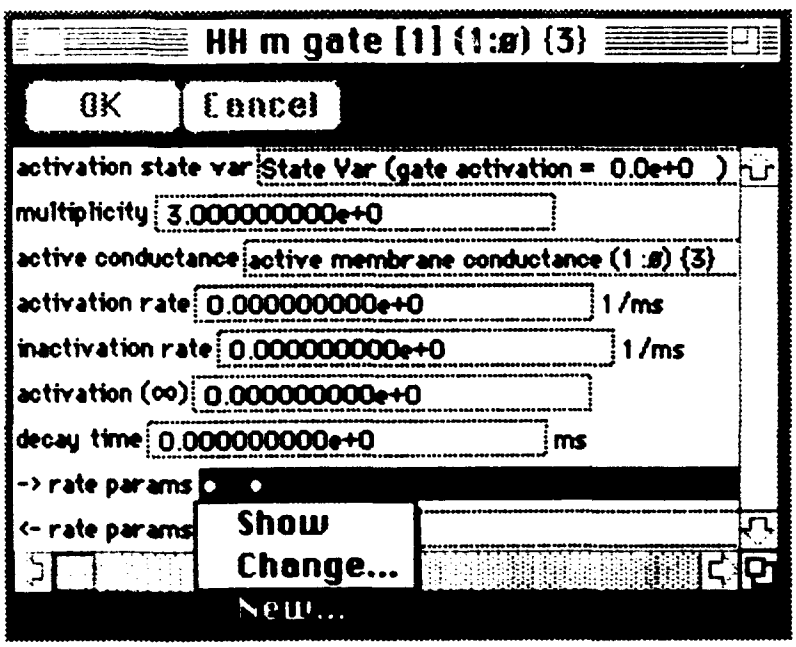
Select *"HH 'm' Gate Activation Rate Function Parameters"* from the *"Make a new"* dialog box and click *"OK"*.

Do the same for the "<- rate params" sub-window, to make a "HH 'm' Gate Inactivation Rate Function Parameters" object.

3.9 Setting the *HH 'm' Gate Activation Rate Function Parameters*

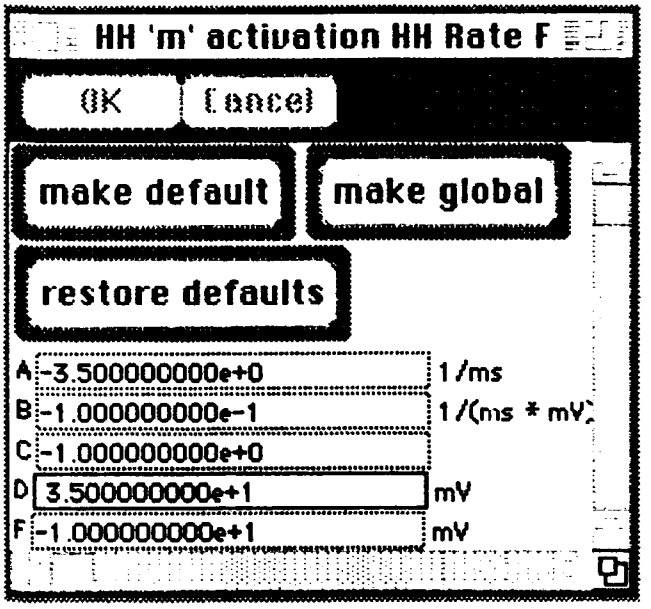
Double click on the text *"HH 'm' Gate Activation Rate Function Parameters"* in the *"-> rate params"* sub-widow (in the *HH m Gate [1] (1:ø) {3}* window):

The instantaneous activation rate, α, is governed by an equation of the form:



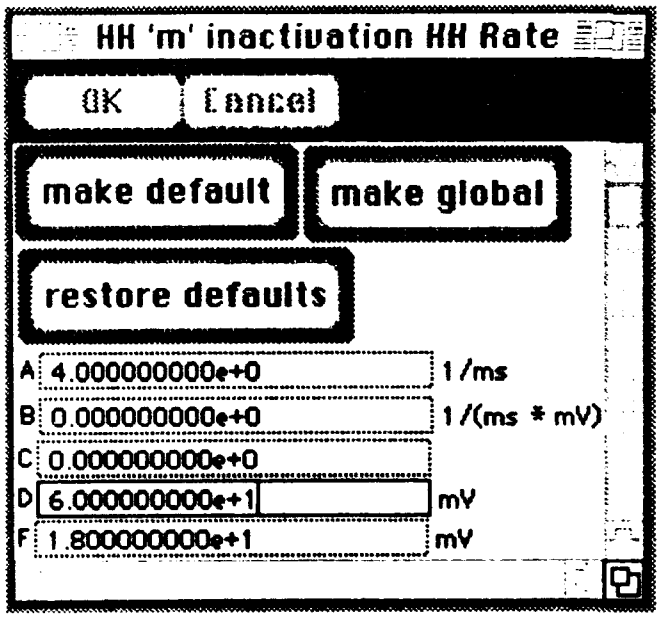
$a(V) = (A + BV)/(C + exp[(V+D)/F]$

Set the parameters *A*, *B*, *C*, *D*, and *F* as shown:



The command button *"make default"* causes this parameter object (and thus these parameters) to be used by all existing *HH m Gates* which do not currently have a parameters object assigned to their *"-> rate params"* sub-widow. The *"make global"* command button makes these parameters universal for all *HH m Gates* regardless of whether or not other parameters have been previously assigned to them. Hitting *"restore defaults"* causes the original parameter values (hard coded into *MacNeuron*) to be restored.

Close the *HH 'm' Gate Activation Rate Function Parameters* window.



3.10 Setting the *HH 'm' Gate Inactivation Rate Function Parameters*

Following 3.9, set the *HH 'm' Gate Inactivation Rate Function Parameters* as shown

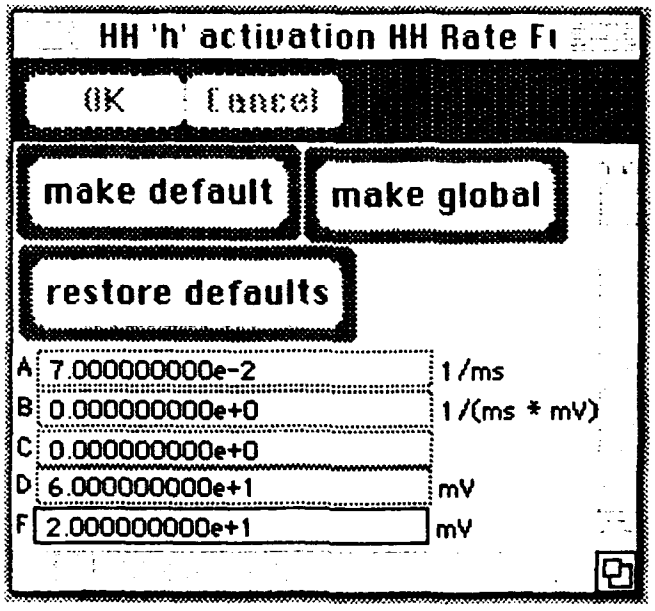
Close the *HH 'm' Gate Inactivation Rate Function Parameters* window.

3.11 Setting the *HH h Gate*

Follow the steps analogous to those in 3.8-3.10 to configure the *HH h Gate [1] (1:ø) {3}.*

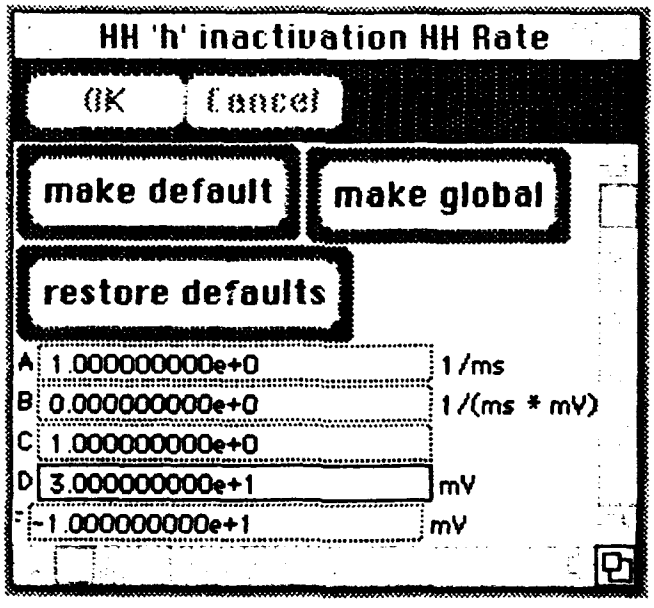
3.12 Setting the *HH 'h' Gate Activation Rate Function Parameters*

Set the *HH 'h' Gate Activation Rate Function Parameters* as shown:



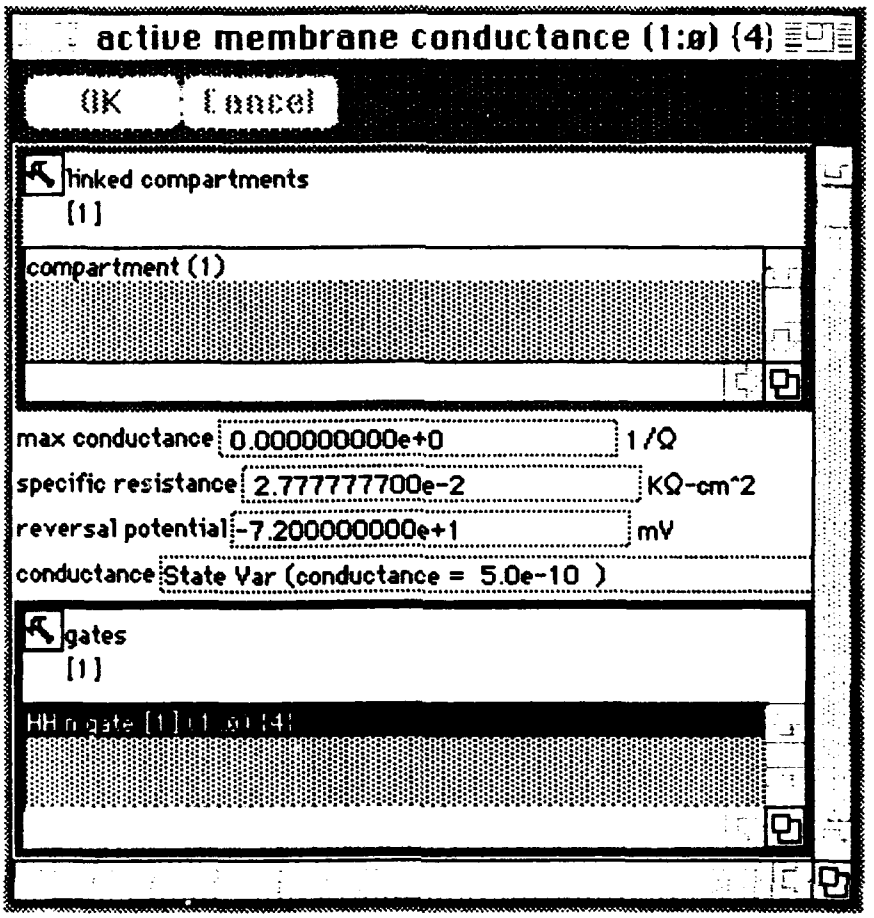
3.13 Setting the *HH 'h' Gate Inactivation Rate Function Parameters*

Set the *HH 'h' Gate Activation Rate Function Parameters* as shown:



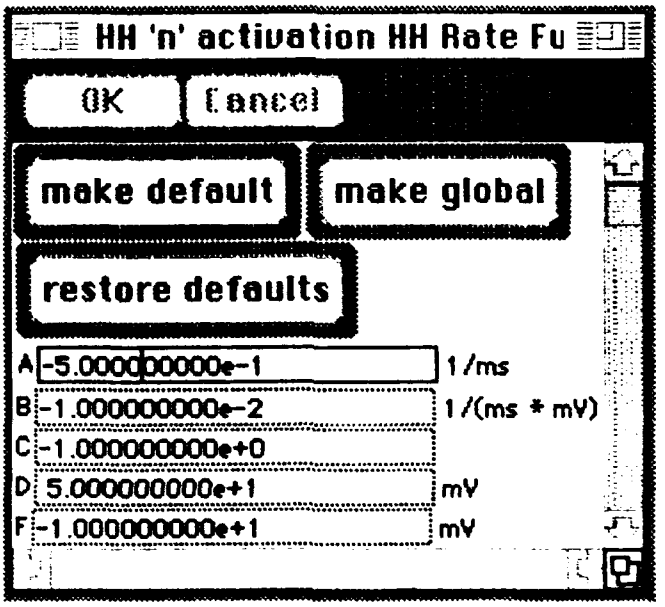
3.14 Setting the *Active Membrane Conductance*: K^+ channel

As in 3.7, configure the *"Active Membrane Conductance (1:ø) {4}"* as follows:



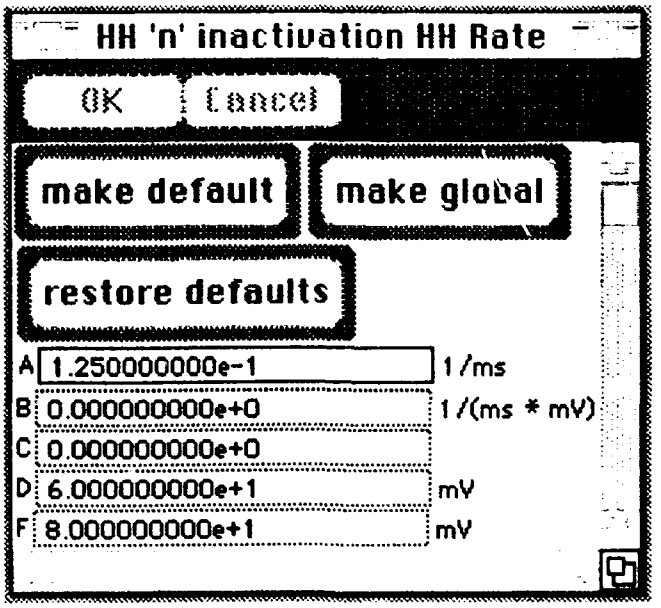
3.15 Setting the *HH 'n' Gate Activation Rate Function Parameters*

Set the *HH 'n' Gate Activation Rate Function Parameters* as shown:



3.16 Setting the *HH 'n' Gate Inactivation Rate Function Parameters*

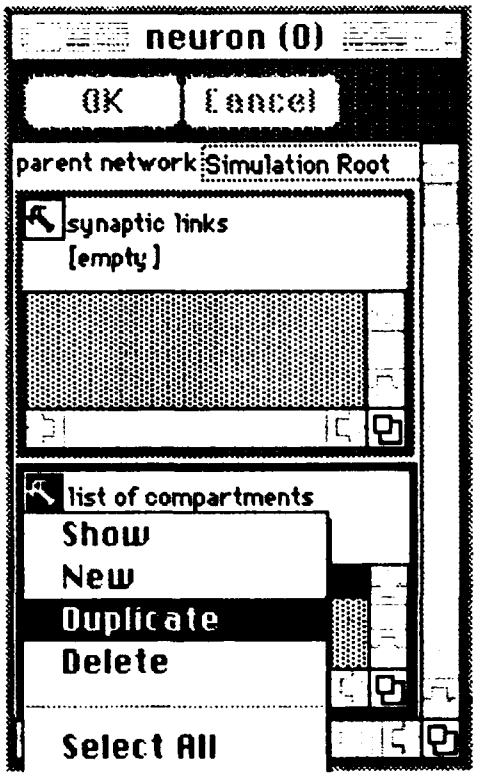
Set the *HH 'n' Gate Inactivation Rate Function Parameters* as shown:



3.17 Duplicating Compartments

We have now built one complete Hodgkin-Huxley compartment, containing a capacitance, a leak membrane resistance, a voltage-gated Na^+ channel, and a voltage-gated K^+ channel. To construct a short length of axonal fiber, we string together two such channels.

Does this mean, you ask in despair, that we have to go through the labor of building another compartment? Fortunately, this is not necessary, because *MacNeuron* has sophisticated built in mechanisms for duplicating objects with complex internal structure. To see this, activatethe root neuron window *(neuron (0))* and select the *compartment (1)* object located in the list of compartments sub-window. An identical compartment can be added to the neuron simply by choosing *"duplicate"* from the tools pull down menu:

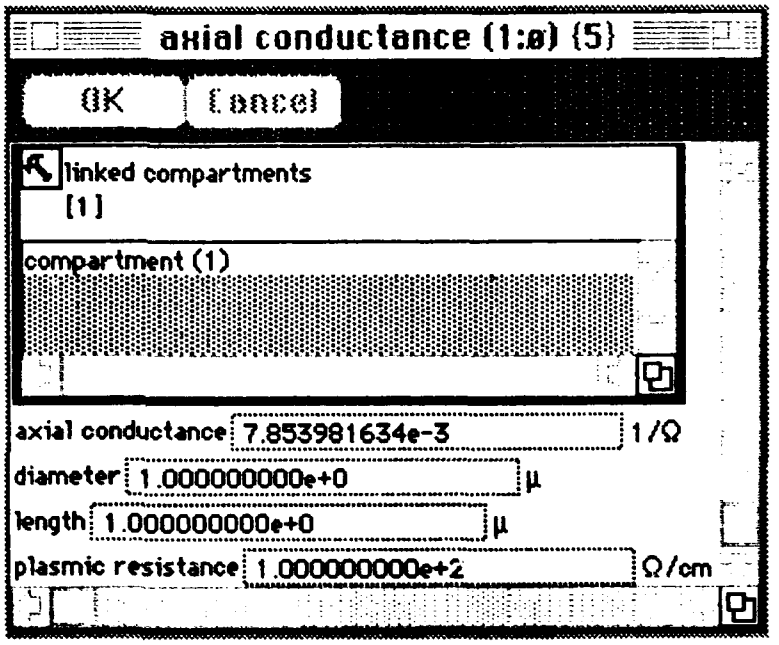


Automatically, a new compartment, identical to the first, is created and added to the root neuron. All of the internal structure of the original is reproduced as well. This is a general feature of *MacNeuron*: Whenever a complex object is duplicated, whether it is a compartment, a neuron, or a network, the entire hierarchy of internal structure is reproduced.

3.18 Chaining Compartments Together with *Axial Conductances*

All we have left to do join the two compartments together.

Activate the window for *compartment (1)* and select *New* from the *compartment links* pull down menu. When the *Make a new* dialog box appears, choose *Axial Conductance* and click *OK* .



Activate the Axial Conductance window:

As it stands, the *Axial Conductance* connects *compartment (1)* to ground. In order to join the other end of the axial conductance to the second compartment, we use a drop in technique.

3.19 Moving Objects to and from Other Objects

Parameter values and objects can be moved to and from other objects easily. All it needs is to hold down the option-key while pressing the mouse button over the selected item. An arrow icon ( ) will appears along with an outline of the selected item. The selected item can then be "dragged", i.e., you can hold down the mouse and move it to the window of another object. "Drop in" the selected item into the appropriate sub-window or editable box and the corresponding changes will be made. If a parameter value is dragged and dropped into another parameter box, that value will be become the new value. Likewise, an object is added to another objects list simply by dropping it in the appropriate sub-window.

Drag *compartment (2)* from the root neuron window over to the *Axial Conductance* window and drop it in the *linked compartments* sub-window. The title of the window should change

to reflect the fact that the *Axial Conductance* now joins *compartment (1)* with *compartment (2)*, and the *compartment (2)* object should now appear in the *linked compartments* sub-window.